MINES ParisTech INRIA

Rapport de stage

Guillaume de Cagny

Du 22 Mars au 27 Août 2010

REMERCIEMENTS

Je tiens à remercier toutes les personnes avec qui j'ai travaillé tout au long de ces quelques mois, notamment Armand Yvet, Cristian Sandu, Sébastien Chaurin, Vincent Ducrot, François Charlot, Mohamed Marouf, Paulo Resende et bien entendu, tout particulièrement mon tuteur de stage Clément Boussard, ainsi que ma tutrice Brigitte d'Andréa-Novel du côté de l'école des Mines.

De façon générale, je remercie l'ensemble de l'équipe IMARA pour ces quelques mois agréables et enrichissants.

Table des matières

1 Introduction **4**
- 1.1 Le contexte . . . 4
  - 1.1.1 L'INRIA et le projet IMARA . . . 4
  - 1.1.2 Les véhicules . . . 6
  - 1.1.3 Les capteurs . . . 8
  - 1.1.4 Un logiciel pour tout assembler : Intempora RTMaps . . . 9
- 1.2 La mission . . . 10
  - 1.2.1 Intérêt d'une telle plate-forme de simulation . . . 10
  - 1.2.2 Le simulateur Pro-SiVIC . . . 11
  - 1.2.3 Contenu de la mission . . . 12

2 Organisation technique et applications de la plate-forme de simulation **14**
- 2.1 Des scripts Pro-SiVIC modulaires . . . 14
  - 2.1.1 Des scripts élémentaires . . . 14
  - 2.1.2 Des scripts scénarios assemblant les précédents . . . 15
  - 2.1.3 Ajout facile de nouveaux scénarios . . . 15
- 2.2 Une forte interchangeabilité . . . 16
  - 2.2.1 SivicLow2High . . . 16
  - 2.2.2 SivicHigh2Low . . . 16
  - 2.2.3 Des composants capteurs ayant la même forme que leurs équivalents réels : . . . 17
- 2.3 Une plate-forme opérationnelle malgré quelques problèmes résiduels 17
  - 2.3.1 Des algorithmes testés et validés . . . 17
  - 2.3.2 Une exploitation plus poussée de la simulation . . . 17
  - 2.3.3 Quelques problèmes résiduels . . . 18

3 Exploitation du simulateur pour améliorer des algorithmes de contrôle **19**
- 3.1 Le suivi de trajectoire . . . 19
- 3.2 Le suivi monotrace . . . 25
  - 3.2.1 Présentation de l'algorithme . . . 25
  - 3.2.2 Intérêt de l'algorithme et perspectives de réalisation . . . 26

3.2.3 fabrication d'un composant RTMaps mettant en oeuvre l'algorithme 27
3.2.4 le déport à l'entrée de virage 27

4 Conclusion **31**

5 Annexe : Mode d'emploi du simulateur **33**

Chapitre 1

Introduction

Ce rapport présente l'intégralité du travail effectué lors de mon stage de fin d'études à l'INRIA Rocquencourt. Dans un premier temps il présentera l'INRIA, et plus particulièrement le projet IMARA dont j'ai fait partie, ainsi que le contexte du stage et les missions qui m'ont été affectées, puis une seconde partie présentera le travail que j'ai effectué. Une troisième partie présentera les résultats obtenus et les utilisations de la plate-forme de simulation que j'ai pu réaliser pour l'INRIA.

1.1 Le contexte

1.1.1 L'INRIA et le projet IMARA

L'INRIA

L'INRIA, Institut National de Recherche en Informatique et Automatique, est un centre national de recherche regroupant les compétences scientifiques publiques dans le domaines des sciences informatique et du contrôle. On lui doit plusieurs projets comme les logicels Scilab et Scicos, concurrents respectivement de Matlab et Simulink, ou encore le langage Caml. Le centre de recherche de Rocquencourt est le premier centre de l'INRIA qui en compte depuis 7 autres partout en France. L'INRIA a fêté ses 40 ans en 2007.

Le projet IMARA

L'équipe IMARA est l'équipe à laquelle j'ai été affecté. IMARA signifiant "Informatique, Mathématiques, et Automatique pour la Route Automatisée", ce projet est celui qui s'occupe de créer la voiture de demain, en équipant le véhicule d'une certaine intelligence pour évacuer tout ou partie de la fonction du conducteur, et ainsi obtenir une conduite plus optimale, moins risquée ou encore de nouvelles formes de transport urbain.

Ce projet fait l'objet d'une JRU (*Joint Research unit*) avec le CAOR de l'école des Mines qui porte le nom de LaRA (La route automatisée). Cette JRU permet de partager du matériel, notamment des véhicules, ainsi que des compétences. Ainsi plusieurs personnes travaillent partiellement à l'école des Mines et à l'INRIA Rocquencourt.

Au sein de l'INRIA, le projet IMARA a la particularité de se démarquer des autres par le fait qu'il se fait pratiquement exclusivement en collaboration avec des entreprises privées ou des administrations, ainsi, plusieurs de ses projets font l'objet de démonstrations ou sont directement financés par des entreprises telles que *Valeo* mais également des structures plus petites telles que la PME *Modulowatt* basée à Paris.

Les projets d'IMARA

Bien entendu, la voiture sans conducteur est un projet très large qui ne peut pas être traité comme un tout, c'est pourquoi l'équipe IMARA se divise en plusieurs sous-projets qui recouvrent la majorité du problème.

- **La communication :** Une des facettes du projet est l'équipe communication, qui se charge de mettre en oeuvre un réseau de communication inter-véhicules ainsi qu'une communication véhicule-infrastructure. Pour ce faire, l'utilisation de technologies vieillissantes est à proscrire, et la mise en oeuvre de technologies naissantes, telles que le protocole IPv6, ou encore d'autres technologies entièrement nouvelles et adaptées à la communication de courte durée, un véhicule ne restant à portée d'un autre véhicule ou d'une portion d'infrastructure que sur une durée très courte.
- **La caméra linéaire :** Le projet de caméra linéaire est un projet de développement de capteur qui permet de déterminer avec précision la position (x, y, θ) d'un émetteur dans le repère du récepteur, ce qui permet dans la pratique de déterminer la position d'un véhicule par rapport à un autre ainsi que son orientation (ce qui permet par exemple de détecter si un véhicule est en train de tourner)
- **Le suivi de véhicules en peloton :** C'est un des projets sur lequel j'ai le plus travaillé. Il s'agit de faire se suivre des véhicules pour former un peloton. La contrainte principale étant d'avoir l'erreur latérale la plus faible possible, c'est à dire que les véhicules doivent reproduire le plus fidèlement possible la trajectoire du précédent, pour ne pas déborder dans les virages par exemple. Ici il s'agit de suivi sans communication, c'est à dire que les véhicules ne communiquent pas entre eux leur position par des moyens de communications informatiques. La détection doit se faire soit au moyen de capteurs embarqués sur un véhicule (tel qu'un télémètre laser) ou sur les deux véhicules (tel que la caméra linéaire)

- **Le SLAM laser :** *(Simultaneous Localization and Mapping)* : Ce projet est maintenant l'un des plus aboutis de l'équipe IMARA. Il consiste au moyen d'un ou plusieurs télémètre(s) laser de détecter la forme de l'environnement d'un véhicule et de créer une carte de cet environnement. Au fur et à mesure que le véhicule se déplace, l'algorithme calcule le déplacement de celui-ci en fonction du déplacement perçu de l'environnement, et en déduit la position actuelle du véhicule par rapport à cet environnement. Cette technologie est particulièrement intéressante de par sa précision (meilleure qu'un GPS), et un fonctionnement possible même en intérieur ou dans des conditions où le GPS ne fonctionne habituellement pas ou mal (montagnes...)
- **Le suivi de trajectoire préenregistrée :** Ce projet est maintenant bien au point et consiste à enregistrer une trajectoire avec un véhicule pour la reproduire plus tard sans conducteur. Ce système est à la base de démonstrations ayant eu lieu notamment en Italie, et sur cette base un système de similaire au métro a pu être présenté : l'utilisateur choisit sa station et le véhicule rejoue les trajectoires préenregistrées jusqu'à la station choisie.
- **Le projet AMARE :** Ce projet, commandé par Modulowatt, consiste à présenter un véhicule électrique qui s'attache automatiquement à une borne (un système qui pourrait s'apparenter au Vélib' présent dans Paris) dans un premier temps puis ensuite d'autre véhicules viendront s'arrimer au premier par le biais d'une attache qui permettra de les recharger électriquement et de se déplacer en peloton.
- **et bien d'autres...** telles que de la détection d'obstacles, de piétons, du Cruise Control (adaptation de la vitesse en fonction du véhicule précédent) etc...

1.1.2 Les véhicules

Le projet IMARA dispose d'un matériel important, notamment au niveau des véhicules :

4 Citroën C3

FIGURE 1.1 – Les véhicules C3.

Appartenant à la Joint Research Unit LaRA, ces voitures ne peuvent pas être modifiées dans leur motorisation mais servent à tester des capteurs et les

équipements de communication. Ainsi, une des C3 est notamment utilisée pour développer l'algorithme de SLAM.

4 CyCabs

Véhicules principaux du projet IMARA, les CyCabs sont des petits véhicules électriques qui servent de plateformes de test ainsi que de démonstrations. Faits sur mesure et adaptés aux activités du projet IMARA, ils ont les caractéristiques suivantes :

- légers : environ 300 kg.
- dimensions : 1,90 m de longueur pour 1,20 m de largeur.
- un CyCab peut accueillir deux personnes avec des bagages.
- motorisation : quatre moteurs électriques pouvant atteindre une vitesse de 30 km/h.
- autonomie : deux heures d'utilisation continue.
- un PC bas niveau sous Linux : ce PC recoit des consignes en vitesse pour chacune des roues, et en braquage des roues, et interagit avec l'électronique pour appliquer ces commandes. A noter que chacune des roues est commandable séparément, en vitesse et en braquage, dans la pratique, cette possibilité n'est pas utilisée et on reste au modèle classique de la voiture avec deux roues arrières droites et une commande appliquée aux roues avant selon le modèle bicyclette.
- un PC haut niveau sous Windows : ce PC est celui qui exécute le scénario désiré : il s'occupe de toutes les interfaces homme/machine, des communications, des capteurs et des algorithmes de traitement qui coordonnent le tout. Il exécute ces scénarios par le biais d'un logiciel de programmation graphique, RTMaps, créé par la société Intempora et qui sera présenté dans la suite de ce rapport.

FIGURE 1.2 – Le CyCab, véhicule phare du projet IMARA.

4 CyBus

Destinés à un usage de démonstration, les CyBus, également dénommés AGV sont construits sur le même principe que les CyCabs mais peuvent accueillir plus de passagers. Ils sont déclinés dans deux modèles, l'AGV ouvert et l'AGV fermé, plus proche d'un véhicule traditionnel.

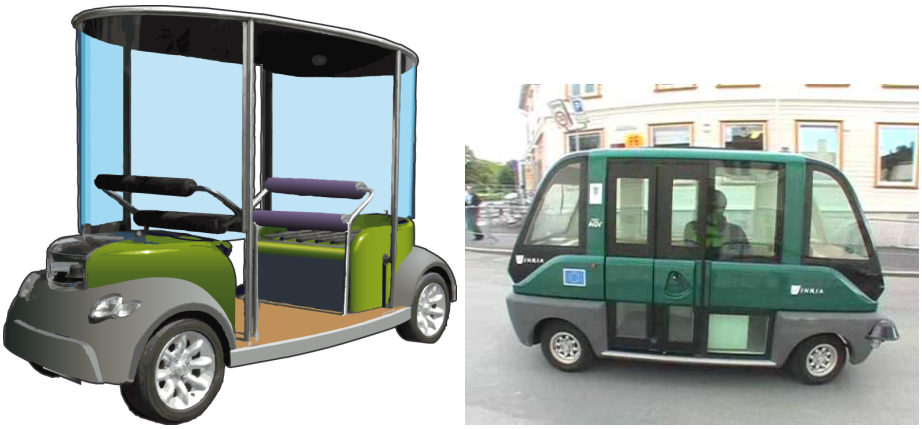


FIGURE 1.3 – Les CyBus, encore appelés AGV.

1.1.3 Les capteurs

Le projet IMARA utilise de nombreux capteurs pour permettre aux véhicules de se répérer dans leur environnement ou d'analyser celui-ci. Cela comprend :

- **Des caméras :** Pour exécuter un traitement d'image et doter le véhicule d'une vision.
- **Des télémètres laser :** Ces télémètres sont constitués d'une diode laser rotative qui envoie un nombre fixé d'impulsions par tour pour déterminer, sur une plage angulaire donnée, la distance à l'obstacle le plus proche dans chaque direction.
- **Des codeurs incrémentaux :** généralement fixés sur les roues des véhicules, ils donnent l'indication du nombre de tours de roues effectués par chaque roue. Selon la qualité du capteur, la précision est de l'ordre d'un 500ème de tour de roue. Certaines techniques permettent de multiplier la précision par 4. Cela permet donc par une technique d'intégration d'obtenir la position du véhicule. Ces capteurs permettent une excellente précision mais ont l'inconvénient d'avoir une erreur qui s'accumule et doivent donc régulièrement être "recalés" par un autre capteur.
- **le GPS - RTK (Global Positioning System - Real Time Kinematic) :** Basé sur un système classique de GPS, le capteur est couplé à une antenne fixe dont la position est connue de façon extrêmement précise et qui reçoit également le signal GPS. De cette façon l'antenne calcule

l'erreur du GPS sur sa propre position et la transmet au véhicule qui peut compenser sa propre erreur de position. Là encore la précision est excellente, mais nécessite d'être à portée de l'antenne RTK.

1.1.4 Un logiciel pour tout assembler : Intempora RTMaps

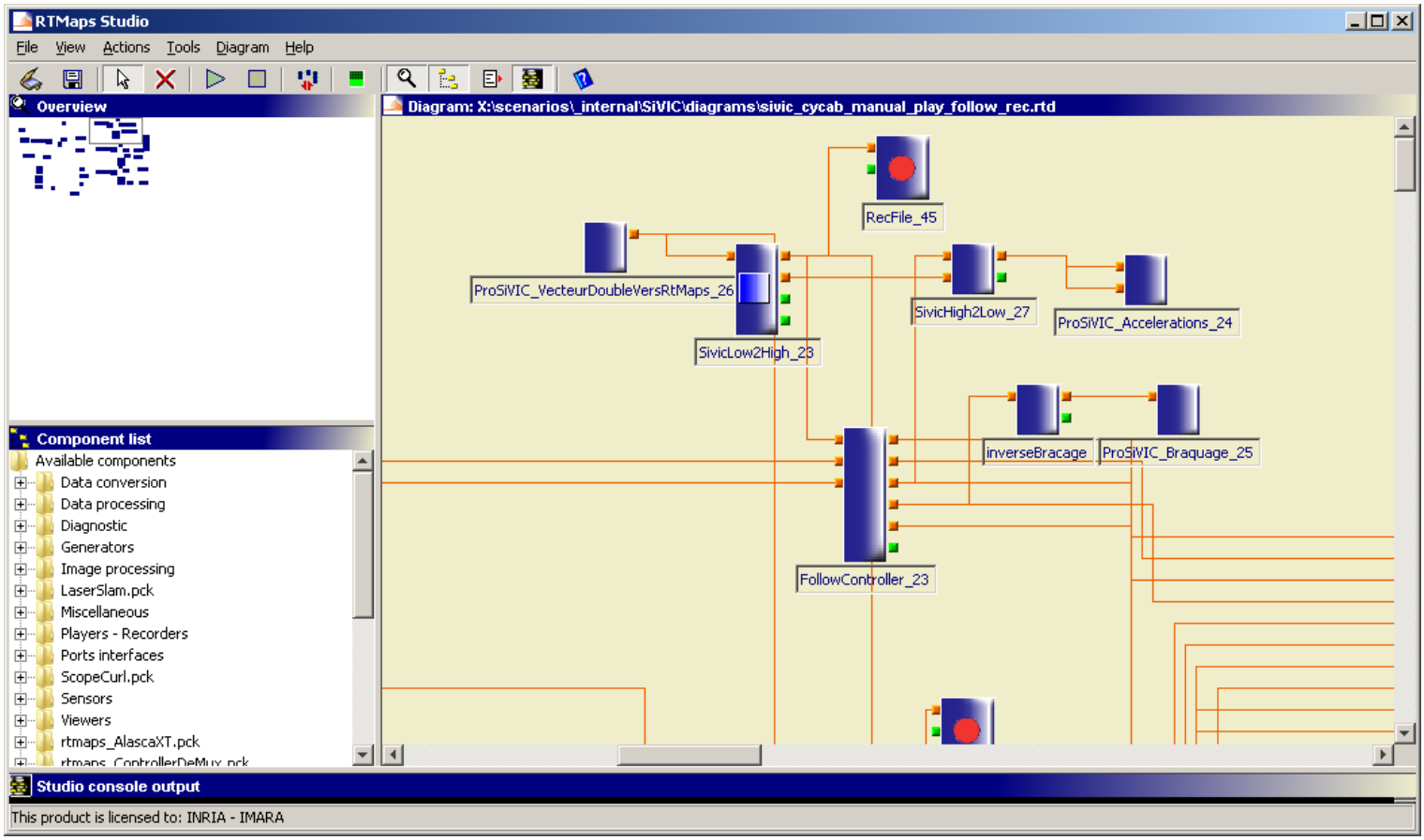


FIGURE 1.4 – Le logiciel RTMaps. Chaque brique bleue est un composant, les lignes oranges sont les liens entre les entrées et sorties de données.

Le logiciel RTMaps est au coeur des scénarios du projet IMARA : c'est le logiciel qui met en oeuvre tous les scénarios. A l'origine créé pour faire de la fusion de données multicapteurs, il met en oeuvre des composants qui sont des exécutables avec des entrées et des sorties de données datées et synchronisées.

Ces composants sont organisés sous la forme d'un diagramme où l'on dispose les composants voulus et l'on connecte les sorties des uns avec les entrées des

autres. Une fois le diagramme créé, chaque composant s'exécute et reçoit des données, les traite et en renvoie. Il n'est pas nécessaire que les composants aient des entrées ou des sorties. Ainsi un générateur d'images n'a pas d'entrée mais une sortie, et un visualiseur d'image n'a pas de sortie mais juste une entrée.

Les composants sont des exécutables (sous Windows, ce sont des librairies DLL) codés en C++ et peuvent donc interagir librement avec le système. RTMaps fournit simplement un "cadre" pour l'exécution et la forme de ces composants. De ce fait, ce logiciel offre des possibilités très larges, puisque l'on peut rajouter n'importe quel composant exécutant n'importe quelle action, et offre également une grande modularité, car tous les composants sont réutilisables et il est très facile de modifier un diagramme pour créer un scénario légèrement différent.

Ce logiciel est omniprésent dans le projet IMARA, ce qui fait que toutes les briques de code développées dans l'équipe sont facilement assemblables en diagrammes et réutilisables, y compris par d'autres. En association avec le système SVN installé dans l'équipe, les composants sont de plus accessibles à tous.

1.2 La mission

L'intitulé de mon stage était "Installation mise en oeuvre d'une plate-forme de simulation physique réaliste pour valider des algorithmes de platooning". Cela a consisté à mettre en oeuvre le logiciel de simulation physique de véhicule Pro-SiVIC et à l'interfacer avec RTMaps de telle sorte qu'il soit facile de passer d'un scénario "réel" à un scénario simulé. De façon à obtenir un moyen simple de valider des algorithmes de contrôle de véhicule ou des algorithmes de traitement de données capteurs. Par la suite, il me fallait implémenter un algorithme de platooning tel que décrit dans un article de Michel Parent et Plamen Petrov, et exploiter les données produites pour pouvoir valider ou non cet algorithme.

1.2.1 Intérêt d'une telle plate-forme de simulation

Malgré le nombre assez important de véhicules dont dispose l'équipe IMARA, tous ne sont pas disponibles en même temps, du fait d'une certaine mobilité des machines situées à l'intérieur, il arrive souvent, même si un véhicule est libre, de ne pas pouvoir l'utiliser car le PC correspondant n'est pas disponible et surtout, il est souvent intéressant pour réaliser des tests de changer la configuration des véhicules, comme la position d'un capteur ou le type de capteur. Ces opérations, qui nécessitent un temps important pour être réalisées en réel, sont pratiquement instantanées en virtuel, puisqu'il s'agit de modifier ou d'ajouter quelques lignes de code dans un script pour pouvoir les exécuter.

De plus, dans la réalité, la validation de la précision de certains algorithmes se heurte à un problème de mesurabilité : il n'est pas possible d'obtenir la position réelle d'un véhicule, seulement de l'estimer quelque soit la méthode utilisée, que ce soit par odométrie, par SLAM, par GPS ou autre. Dans le logiciel de simulation, il est possible d'accéder à un nombre important de valeurs "exactes"

ce qui permet de comparer facilement des algorithmes de traitement de données et de mesurer leur erreur, dans le cas où l'oeil nu ne permet pas d'apprécier cette précision.

Par ailleurs, le logiciel Pro-SiVIC étant conçu pour fonctionner en parallèle avec RTMaps (il lui transmet toutes ses informations et peut recevoir des commandes d'RTMaps) il est un parallèle tout-à-fait naturel avec le matériel de l'INRIA, c'est pourquoi j'ai essayé d'exploiter au maximum cette possibilité afin de pouvoir interchanger les éléments simulés sous Pro-SiVIC avec les véritables véhicules / capteurs de l'INRIA, notamment en leur donnant le même format pour les entrées et sorties.

Enfin, signalons qu'un tel simulateur offre des possibilités importantes de démonstration. Si le déplacement d'un véhicule et l'organisation d'une véritable démonstration relève d'un projet à part entière, la démonstration d'une simulation est d'une simplicité presque enfantine dans la mesure où il est possible d'enregistrer des vidéos et que la seule chose à transporter est un fichier. Dans le cas où l'on souhaiterait faire une démonstration du simulateur en conservant les possibilités d'interaction et ne pas se contenter d'une vidéo "figée", il suffirait alors de déplacer l'ordinateur constituant la plate-forme de simulation pour pouvoir établir une démonstration.

1.2.2 Le simulateur Pro-SiVIC

Le logiciel Pro-SiVIC est issu du logiciel Sivic, développé conjointement par plusieurs laboratoires de recherche pour la plupart partenaires de l'INRIA : le Laboratoire Central des Ponts et Chaussées (LCPC), l'INRETS et le Livic. A l'origine un moteur open-source, ce logiciel est maintenant vendu par la start-up Civitec, qui a construit autour du concept original une nouvelle interface basée sur une fenêtre Windows où l'on peut régler les paramètres sans passer par la console.

Sivic est un moteur physique et graphique permettant de simuler de façon réaliste des véhicules dans un environnement. Le moteur intègre un modèle dynamique de véhicule poussé, qui tient compte dans la simulation de paramètres complexes tels que les amortisseurs, le rayon sous charge des roues ou encore le glissement.

En outre l'intérêt principal de Sivic est de proposer différents modèles de capteurs reprenant la plupart des capteurs utilisés dans le projet IMARA : télémètres laser, caméras (pouvant être utilisées soit pour visualiser la scène, soit en tant que capteurs, auquel cas on peut leur ajouter des filtres de bruit, de flou...), codeurs incrémentaux pour l'odométrie...

Ce logiciel est particulièrement adapté à Imara dans la mesure où il est conçu pour fonctionner avec RTMaps : toutes les données capteur sont exploitées par RTMaps tandis que celui ci retransmet une commande de couple à appliquer au véhicule. Ce transfert se fait au moyen de composants RTMaps fournis avec le logiciel.

FIGURE 1.5 – Le logiciel Pro-SiVIC.

1.2.3 Contenu de la mission

Mon stage s'est déroulé en trois parties :

Installation et prise en main du logiciel SiVIC.

La première partie de mon stage dans le projet Imara a été de mettre en place le simulateur sur une machine acquise à cet effet. Disposant d'une configuration puissante, celle-ci était censée permettre à l'INRIA de faire fonctionner le simulateur de façon immersive, avec trois écrans, et des périphériques d'interaction simulant les véhicules : volant, joystick, de façon à obtenir une plate-forme de démonstration.

Passé cette installation, il fallait ajouter au simulateur les véhicules IMARA. Il s'agissait donc d'implanter dans le simulateur des modèles 3D des véhicules utilisés à l'INRIA, notamment le CyCab, et de les paramétrer comme ceux-ci : rayon des roues, empattement, poids... afin d'obtenir un comportement de véhicule le plus proche possible des véhicules réels.

Adaptation du simulateur au projet IMARA.

Une fois le simulateur installé, il fallait adapter son fonctionnement à celui des projets de l'INRIA, c'est à dire faire en sorte que des programmes, notamment des diagrammes RTmaps, conçus pour réaliser des scénarios sur véhicules puissent être réutilisés avec un minimum d'effort pour faire des simulations SiVIC. Cela s'est traduit par l'écriture en C++ de plusieurs composants pour RTMaps :

- un pour pouvoir commander le véhicule avec une commande en vitesse comme un véritable CyCab.
- un pour récupérer les informations de la voiture et les formater au même format que les données sortant du PC bas niveau d'un CyCab réel.
- un pour pouvoir obtenir une sensation immersive avec l'affichage sur trois écrans et la possibilité de changer de caméra en cliquant sur l'écran.

Utilisation du simulateur et exploitation de données

Une fois la plateforme de simulation prête et opérationnelle, j'ai pu m'en servir pour étudier et essayer d'améliorer des algorithmes de contrôle. Plusieurs algorithmes ont pu être essayés, soit à des fins de démonstration, soit à des fins de déboguage, mais deux ont été étudiés dans leur fonctionnement et ont pu être travaillés :

- l'algorithme de suivi d'une trajectoire préenregistrée.
- l'algorithme de commande en peloton correspondant à la caméra linéaire.

Dans la section suivante, nous allons rentrer dans les détails techniques de mon travail sur la plate-forme de simulation.

Chapitre 2

Organisation technique et applications de la plate-forme de simulation

Comme dit précédemment, la plate-forme de simulation est constituée d'un cadre logiciel (Pro-SiVIC / RTMaps) et d'un certain nombre de petits outils permettant de substituer les programmes entre un véritable véhicule et un scénario simulé. Je vais maintenant présenter tous ces outils que j'ai pu réaliser pour rendre l'utilisation de la plate-forme la plus simple possible.

2.1 Des scripts Pro-SiVIC modulaires

Basiquement, Pro-SiVIC fonctionne avec une ligne de commande qui permet de lui ajouter des objets, de régler leur propriétés, d'appeler leurs fonctions. Pour obtenir un scénario, il faut regrouper toutes les commandes créant le scénario (création de voiture, choix du modèle, réglage des paramètres...) dans un fichier texte appelé script. Il me fallait écrire tous les scripts permettant d'obtenir les scénarios usuels, et notamment ceux qui m'ont servi par la suite, à savoir le suivi de trajectoire, le suivi monotrace et d'autres.

2.1.1 Des scripts élémentaires

Pour que ces scripts soit réutilisables au maximum, ils ont été créés de façon modulaire, c'est à dire divisés en petits scripts. En effet, comme il existe une commande qui exécute toutes les commandes d'un script (`run fichier.script`) il est facile de diviser le code en plusieurs morceaux réutilisables. Ainsi de nombreux scripts élémentaires ont été créés :

- des scripts environnement, permettant de choisir entre un environnement vide (plus léger) ou urbain.
- 4 scripts créant une voiture, un CyBus ou un CyCab
- 4 scripts ajoutant un télémètre laser à la voiture concernée.
- des scripts ajoutant des odomètres à chacun des véhicules. Ces odomètres ont été réglés pour le projet AMARE et contiennent de cette façon le même nombre de ticks par tour de roue que ceux utilisés dans ce projet.
- des caméras
- des observeurs. Ces composants sont spécifiques au simulateur SiVIC et permettent de transmettre toutes les données concernant un véhicule : rayon sous charge des roues, vitesse instantanée de chaque roue, mais aussi les valeurs "exactes" de paramètres plus usuels tels que la position ou la vitesse du véhicule.

Ensuite, tous ces scripts ont été réutilisés dans des scripts correspondants cette fois à des scénarios, qui peuvent être mis en place et directement utilisés.

2.1.2 Des scripts scénarios assemblant les précédents

Voici une liste non exhaustive des scénarios créés :

- Scenarios de suivi de trajectoire, en enregistrement et en rejeu, comportant un unique véhicule, en environnement urbain ou vide.
- Scenarios de suivi monotrace : environnement urbain ou vide, avec deux ou quatre véhicules.
- Scenario de SLAM : celui-ci a besoin par nature de l'environnement urbain puisque le SLAM doit en dresser la carte.
- Scenario de parking automatique : celui-ci a été créé pour tester des algorithmes de créneau automatique.
- Scenario comportant une voiture avec des odomètres, qui a été utilisé pour AMARE pour comparé la position calculée du véhicule avec sa position réelle.
- ...

2.1.3 Ajout facile de nouveaux scénarios

La liste précédente laisse transparaître qu'il est facile, avec les scripts élémentaires existants, de créer un scénario ou de modifier un existant : pour rajouter un véhicule ou le modifier, changer d'environnement, ajouter un capteur etc..., il suffit d'appeler le script correspondant à la place de celui qu'on veut remplacer. Cela se traduit par la modification de quelques lignes de code, ce qui à comparer avec la modification d'un véhicule réel permet une économie de temps importante. On se reportera pour les détails techniques au manuel en annexe de ce rapport.

2.2 Une forte interchangeabilité

Comme précisé, on souhaite pouvoir échanger facilement un diagramme simulé avec son équivalent réel. Pour cela il a fallu mettre en place deux composants :

2.2.1 SivicLow2High

Ce composant est conçu pour être échangé avec un composant utilisé à l'INRIA dans les CyCabs et qui est dénommé CycabLow2High. Comme son nom le suggère, ce composant sert à récupérer l'information du PC bas niveau, notamment la position et la vitesse du CyCab. Assez naturellement, ce composant récupère les informations sur le véhicule virtuel (à l'aide d'un observeur) et les formate pour avoir les mêmes sorties que CycabLow2High.

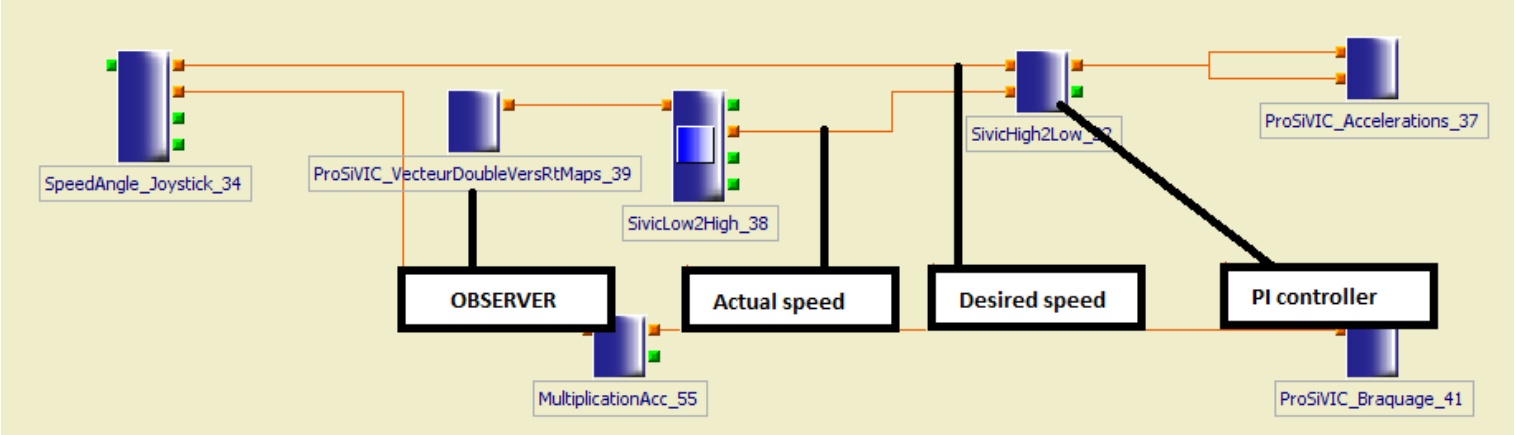


FIGURE 2.1 – Le contrôleur PI, constitué de deux blocs RTMaps.

2.2.2 SivicHigh2Low

Ce composant, de la même façon, est conçu pour remplacer CyCabHigh2Low, à savoir qu'il transmet une consigne de vitesse au véhicule virtuel. Cela pose un léger problème dans la mesure où un CyCab réel est commandé en vitesse, alors qu'un véhicule de SiVIC est commandé en couple. Dans cette mesure, il a fallu implémenter un contrôleur pour obtenir la bonne valeur de la vitesse dans un temps raisonnable. Il a été choisi d'utiliser un contrôleur classique de type PI. (Proportionnel-Intégral). Pour que ce contrôleur fonctionne, il est donc nécessaire d'avoir un feedback sur la vitesse réelle du véhicule. Ce retour est bien sûr pris sur les sorties de SivicLow2High. Ce retour est la seule différence entre les modules utilisés sur les scénarios réels, dans la mesure où l'utilisation de SivicHigh2Low requiert celle de SivicLow2High, chose qui n'est pas nécessaire sur les véritables CyCabs.

Ces deux composants m'ont permis de trouver des équivalents à l'interface entre PC bas niveau et haut niveau. De la même façon, des composants SiVIC correspondant aux différents capteurs donnent les mêmes données que leurs équivalents "réels" :

2.2.3 Des composants capteurs ayant la même forme que leurs équivalents réels :

Les composants suivants, inclus dans Pro-SiVIC, ont une forme directement analogue à celle de leurs équivalents réels.

- ProSiVIC_Telemetre s'échange avec AlascaXT qui est le composant qui renvoie les informations du laser Alasca utilisé à l'INRIA
- ProSiVIC_Odometre renvoie les informations sur la distance parcourue, tel les codeurs incrémentaux. Cette information est équivalente à un nombre de ticks et peut être transformée à l'aide d'un simple calcul.
- ProSiVIC_View renvoie une image, qui peut être traitée de la même façon qu'un capteur caméra. A noter qu'il y a un bien sur un risque que les algorithmes de vision se comportent différemment sur une image simulée, bien moins complexe qu'une image réelle.

2.3 Une plate-forme opérationnelle malgré quelques problèmes résiduels

En terme de résultats, la plate-forme de simulation est pleinement opérationnelle. Il est tout à fait possible de créer n'importe quel scénario et de le connecter à un diagramme RTMaps, donc d'expérimenter des algorithmes de contrôle ou de traitement de données capteur, sur un environnement virtuel donc sans risque et sans erreur de mesure.

2.3.1 Des algorithmes testés et validés

Plusieurs algorithmes ont été expérimentés à l'aide de la plate-forme de simulation :

- le SLAM, qui était déjà au point et a montré une bonne performance
- le parking automatique, dont la simulation a permis de corriger les bugs.
- le suivi de trajectoire, qui a pu être amélioré.
- le suivi monotrace pour la conduite en peloton, dont un problème a pu être mis en évidence, et pour lequel des tentatives d'améliorations ont pu être faites.

2.3.2 Une exploitation plus poussée de la simulation

Un autre des avantages de la simulation est que chaque donnée peut être exploitée. Ainsi, il est possible d'enregistrer les trajectoires des véhicules, ce qui permet ensuite de les comparer dans le cas du suivi de trajectoire ou du

suivi monotrace. Dans le cas du SLAM, la simulation pourrait être exploitée en enregistrant les positions réelles du véhicules et celles données par le SLAM, ce qui permettrait de vérifier la précision de la localisation.

Par ailleurs, il est également possible de produire des vidéos des simulations, ce qui a été fait pour les quatre exemples cités ci-dessus. De telles vidéos ont un intérêt indéniable de démonstration ou de présentation des travaux de l'INRIA.

2.3.3 Quelques problèmes résiduels

Dû à des problèmes de performances graphiques de Pro-SiVIC, il n'est pas possible d'obtenir une simulation satisfaisante en temps réels sur les trois écrans, comme ce qui était prévu au départ. En effet, l'utilisation systématique d'effets graphiques sur tout l'environnement demande une puissance de calcul colossale que même les dernières générations de processeurs graphiques n'ont pas pu offrir. Cette fonctionnalité devra donc attendre une optimisation du moteur de Pro-SiVIC.

D'autre part, le réglage des amortisseurs s'est avéré difficile, malgré que le modèle soit présent dans le manuel de Pro-SiVIC et on observe toujours un tangage excessif des véhicules. Ce tangage est gênant dans la plupart des simulations qui utilisent un laser, car les données de celui-ci sont erronées car celui-ci voit le sol lorsque le véhicule penche vers l'avant. Un réglage plus fin de ces amortisseurs virtuels, basé sur une étude de l'amortisseur du CyCab réel, serait souhaitable pour obtenir un résultat plus fidèle à la réalité.

La partie suivante présentera plus en détail les résultats de recherche obtenus sur les algorithmes de suivi de trajectoire et de suivi monotrace.

Chapitre 3

Exploitation du simulateur pour améliorer des algorithmes de contrôle

3.1 Le suivi de trajectoire

Cette section concerne l'amélioration du suivi de trajectoire préenregistrée par le contrôleur RoutePlanner de l'INRIA. Ce contrôleur prend en effet un fichier en entrée, ainsi qu'un point à atteindre et calcule la trajectoire à parcourir à partir d'une suite de points.

La commande du CyCab se fait de façon classique avec une vitesse et un angle de braquage. Ici, on applique une vitesse de référence arbitraire. La commande appliquée en braquage est un correcteur proportionnel de l'erreur latérale et de l'erreur en angle du type

$$\beta = K_\perp \cdot e_\perp + K_\theta \cdot e_\theta \tag{3.1}$$

où $e_\perp$ est l'écart latéral avec la trajectoire et e_θ l'erreur d'orientation du véhicule.

Il est apparu que ce type de contrôle perdait beaucoup d'information en se contentant d'une correction d'erreur. Il est beaucoup plus judicieux d'inclure un angle volant de référence, que l'on peut soit enregistrer pendant l'enregistrement de la trajectoire, soit calculer ensuite en fonction de la courbure, ce qui nous donne une commande en

$$\beta = \beta_{ref} + K_\perp \cdot e_\perp + K_\theta \cdot e_\theta \tag{3.2}$$

Pour valider cette amélioration, nous utilisons le simulateur SiVIC pour enregistrer une trajectoire et la rejouer. Les graphiques des trajectoires suivies avec les différentes méthodes. En traçant les trajectoires qu'on se rend compte que sans le β_{ref} on observe effectivement un retard aux virages et parfois un phénomène d'oscillation.

En rouge, sont tracées les trajectoires du véhicule qui enregistre. En vert, les trajectoires reproduites par le contrôleur. Deux trajectoires sont effectuées, puis présentées d'abord sans angle de braquage préenregistré (en correction d'erreur) puis avec l'amélioration qui consiste à rajouter un angle de braquage de référence.

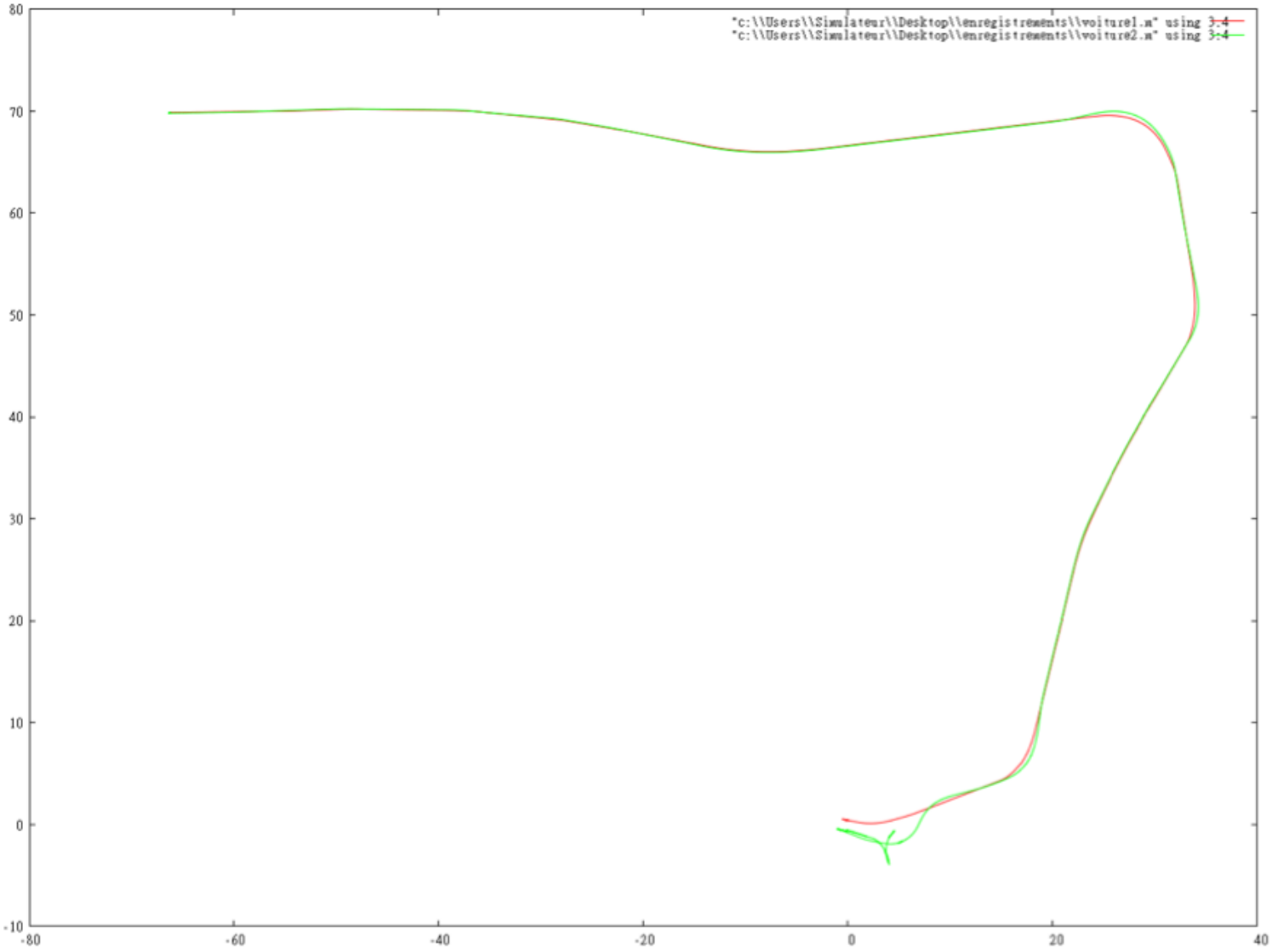


FIGURE 3.1 – Sans l'angle de braquage.

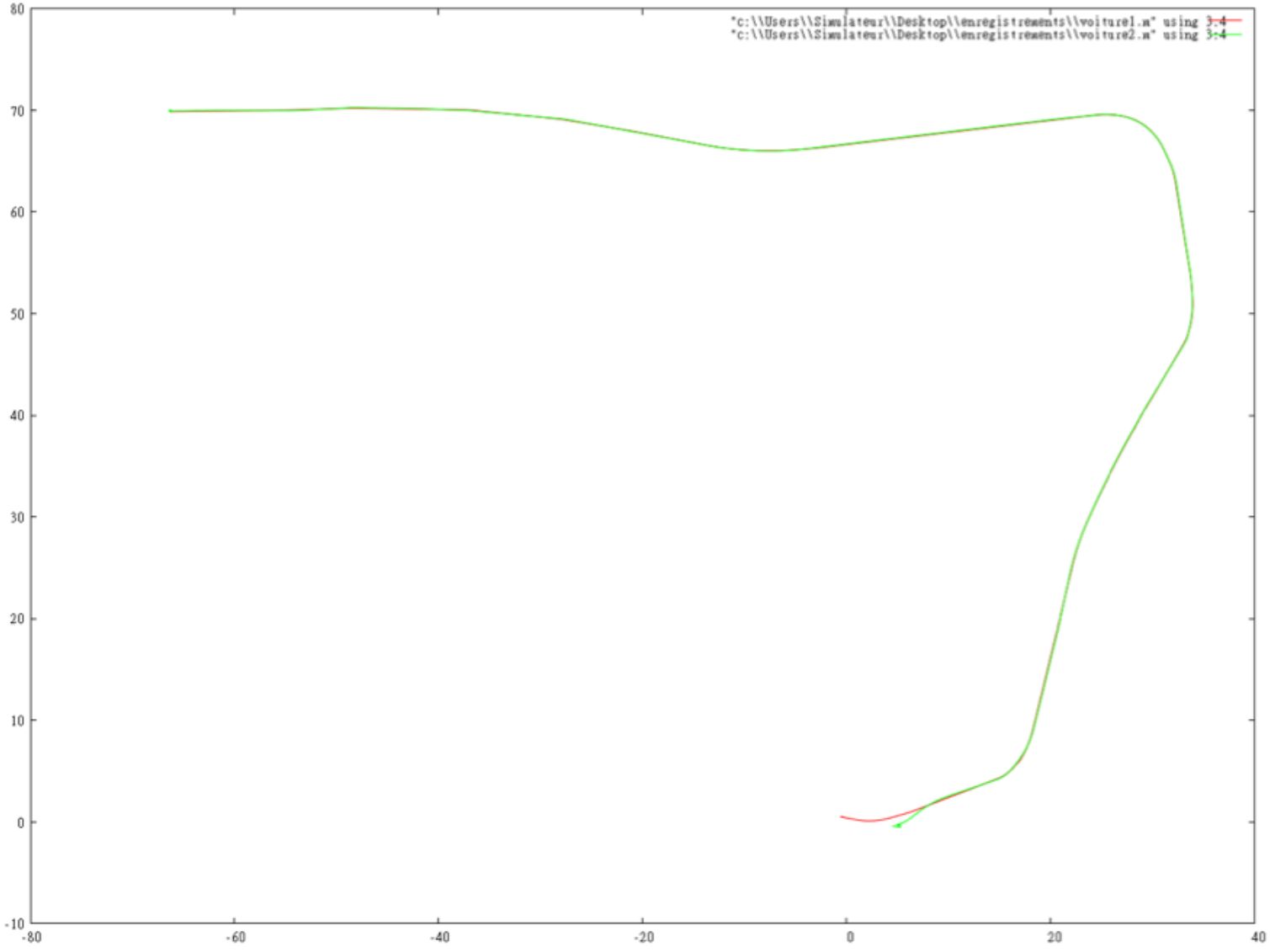


FIGURE 3.2 – Avec l'angle de braquage.

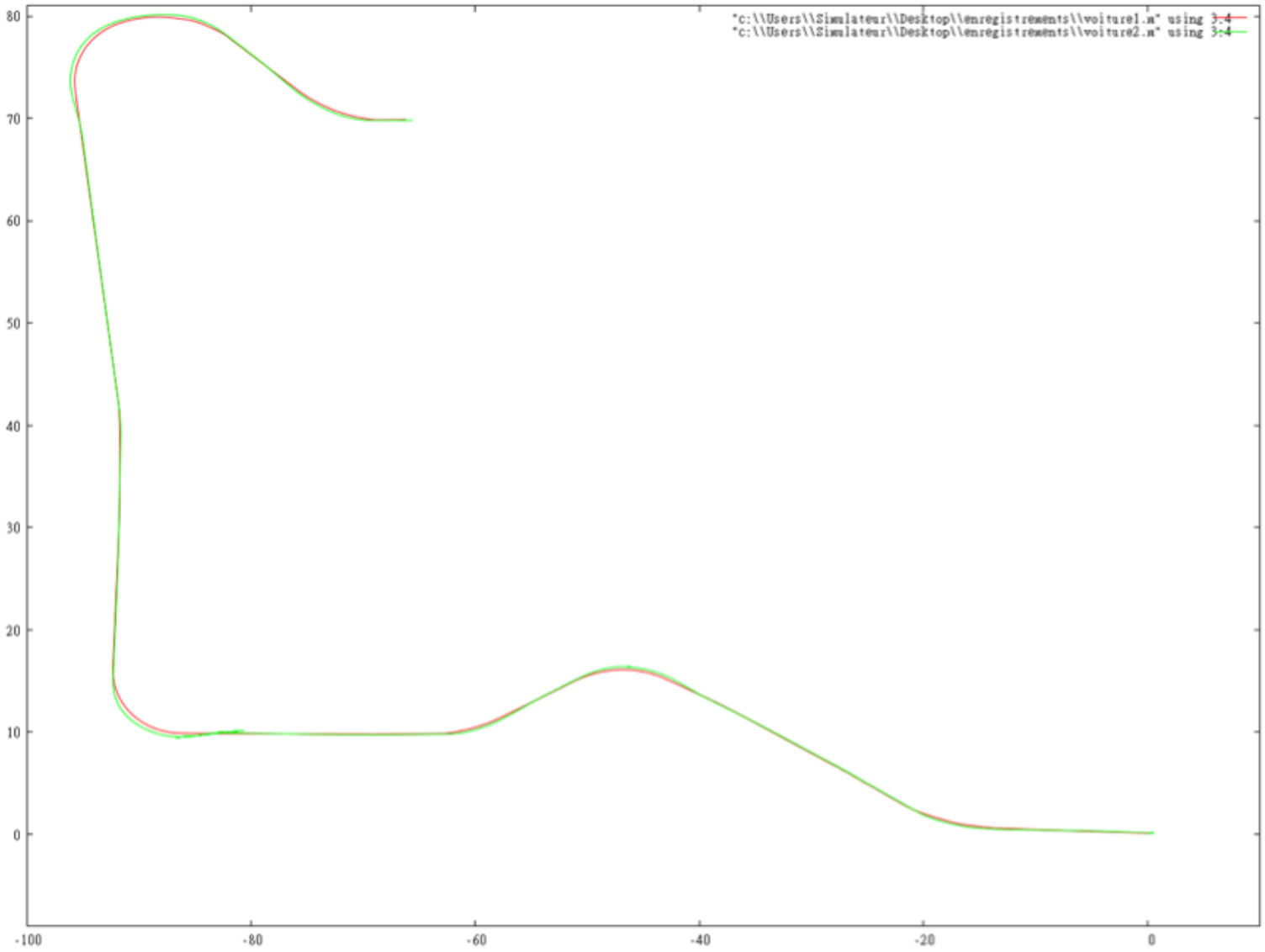


FIGURE 3.3 – Sans l'angle de braquage.

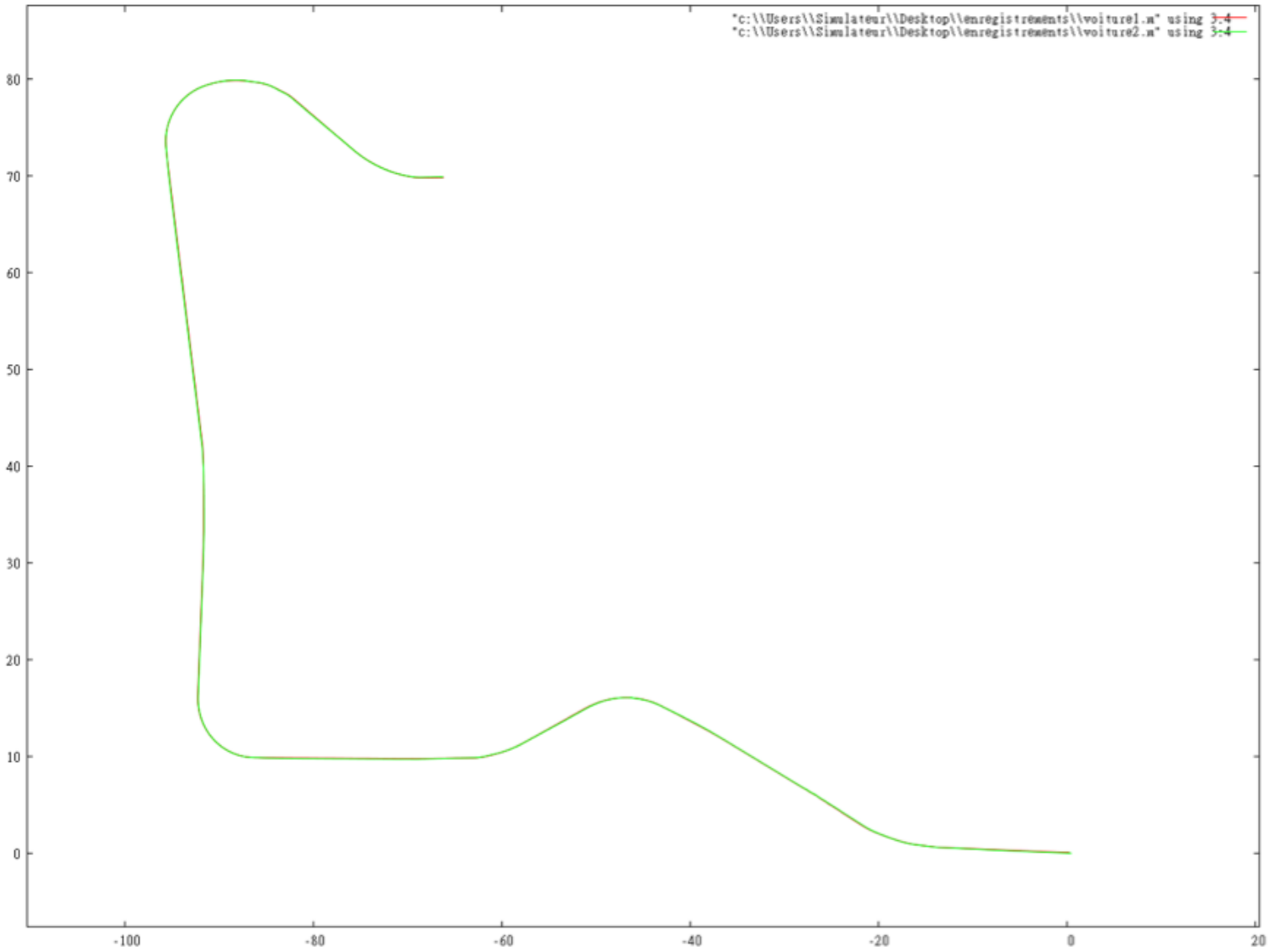


FIGURE 3.4 – Avec l'angle de braquage.

On constate bien une amélioration du suivi de trajectoire. Dans ce cas précis, il serait sans doute facile de montrer mathématiquement que la seconde méthode est meilleure que la première. En revanche, pour des situations plus complexes, l'intérêt de la simulation apparaît de façon claire, puisqu'une simple simulation renseigne l'utilisateur sur l'intérêt d'un algorithme par rapport à un autre. Dans le cas où ça se révèle concluant, il peut être intéressant d'approfondir les mathématiques sous-jacentes, dans le cas contraire, cela permet de se rendre compte instantanément que la piste suivie n'est pas la bonne. L'utilisation du simulateur résulte donc encore une fois dans un gain de temps.

3.2 Le suivi monotrace

Je vais maintenant présenter un algorithme que j'ai pu étudier d'après l'article de Michel Parent et Plamen Petrov, "Nonlinear Adaptive Control for Autonomous Vehicle Following". Cet article présente une méthode de suivi qui permet d'obtenir, sous certaines conditions, le suivi monotrace.

3.2.1 Présentation de l'algorithme

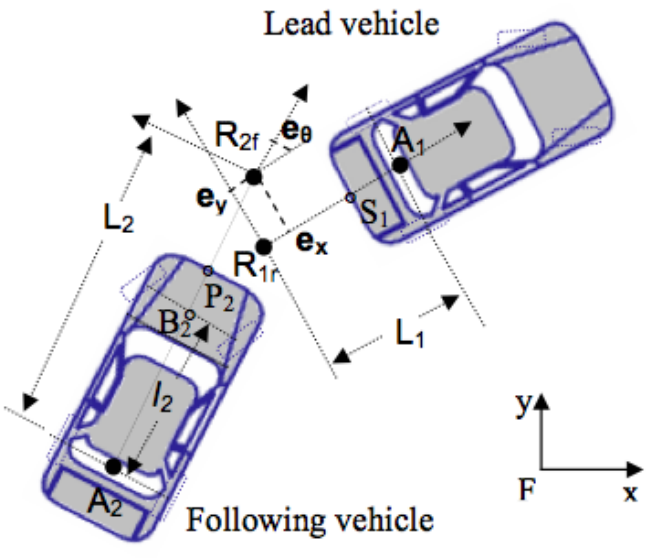


FIGURE 3.5 – Algorithme de suivi monotrace.

Il s'agit, plutôt que de recourir à un asservissement de la distance autour d'une valeur fixée - ce qui pose des problèmes de "cutting the corner" -, d'asservir la position d'un point fictif de la voiture avant sur celle d'un autre point fictif

de la voiture arrière. En l'occurrence, on choisit un point en arrière du véhicule avant, le point R_{1r}, et un point en avant du véhicule arrière, le point R_{2f}.

On asservit alors la position de R_{2f} sur celle de R_{1r}, et si L_1 et L_2 sont les longueurs de la figure, alors, pour $L_1 = L_2$ le suivi monotrace sera obtenu en courbe stable. Par courbe stable, on entend lorsque le rayon de courbure est constant et établi.

Présentons maintenant l'algorithme en détail : on se place dans le repère local du véhicule de tête. On y calcule les équations différentielles régissant les erreurs : e_x erreur en distance, e_y erreur latérale, et $e_\theta = \theta_2 - \theta_1$ (attention e_θ n'est pas une erreur au sens où on ne cherche pas à l'annuler : en courbe, cette valeur n'est pas nulle.)

Ces equations sont :

$$\begin{bmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\theta} \end{bmatrix} = \begin{bmatrix} \cos e_\theta & -L_2 \sin e_\theta \\ \sin e_\theta & L_2 \cos e_\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & -L_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ \omega_1 \end{bmatrix} + \dot{\theta_2} \begin{bmatrix} e_y \\ -e_x \\ 0 \end{bmatrix} \tag{3.3}$$

où (v_1, ω_1) et (v_2, ω_2) sont les vitesses linéaires et angulaires des véhicules 1 et 2.

on opère un changement de commande, et on pose deux commandes u_1 et u_2 telles que :

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos e_\theta & -L_2 \sin e_\theta \\ \sin e_\theta & L_2 \cos e_\theta \end{bmatrix} \begin{bmatrix} v_2 \\ \omega_2 \end{bmatrix} \tag{3.4}$$

On suppose que L_2 est non nul (ce cas ne correspond de toute façon pas à une situation réelle), ce qui nous permet d'assurer que l'on pourra passer d'une commande à l'autre. On obtient

$$\begin{aligned} \dot{e_x} &= -v_1 + \omega_1 e_y + u_1 \\ \dot{e_y} &= (L_1 - e_x)\omega_1 + u_2 \end{aligned} \tag{3.5}$$

On pose alors :

$$\begin{aligned} u_1 &= -k_x e_x + v_1 - \omega_1 e_y \\ u_2 &= -k_y e_y - (L_1 - e_x)\omega_1 \end{aligned} \tag{3.6}$$

Ce contrôleur permet d'asservir la position du point fictif du véhicule arrière sur celui du véhicule avant, ce qui répond en théorie au problème. Cependant, il reste à estimer les vitesses v_1 et ω_1, ce qui correspond à la partie adaptative du contrôleur. Cette partie n'ayant pas été mise en place dans la simulation, je ne la décris pas ici.

3.2.2 Intérêt de l'algorithme et perspectives de réalisation

Obtenir un suivi monotrace simple est une chose très intéressante pour mettre en place un accrochage immatériel entre plusieurs véhicules. Ainsi dans

le cadre du projet AMARE, qui utilise lui une accroche matérielle, un tel algorithme pourrait permettre de minimiser l'effort sur l'accroche, à condition de ne pas avoir trop d'erreur dans la position relative des points fictifs. En l'occurrence, cet algorithme a été mis en place pour tester la mise des véhicules en peloton.

Cet algorithme a pour intérêt majeur de ne nécessiter pour sa mise en place que très peu d'information : seules la position et l'orientation du véhicule de tête dans le repère du véhicule suiveur sont requises. Or, ces informations sont exactement celles qui sont données par la caméra linéaire, c'est pourquoi il apparaît très intéressant de développer cet algorithme avec ce capteur.

3.2.3 fabrication d'un composant RTMaps mettant en oeuvre l'algorithme

Dans la mesure où l'algorithme a besoin d'informations sur la distance et l'orientation du véhicule suiveur dans le repère du leader, et dans la mesure où le capteur caméra linéaire n'est pas présent dans SiVIC ces informations ont été récupérées dans les observeurs de SiVIC. Ce qui constitue une facilité, puisqu'on récupère position et orientations réelles de chaque véhicule, avant de calculer les informations qui nous intéressent. Cela évite les erreurs de mesure et de capteurs. En revanche pour implémenter cet algorithme sur véhicule il faudra comprendre le fonctionnement des estimées et les implémenter.

Les premiers résultats de simulation confirment le bon fonctionnement de l'algorithme. on constate bien un suivi monotrace dans les courbes stables.

3.2.4 le déport à l'entrée de virage

Il y a toutefois un phénomène de déport en entrée de virage : le véhicule suiveur commence par faire un déport dans la direction opposée du virage du véhicule de tête. L'exploitation des données de simulation ont clairement fait apparaître ce déport de façon plus claire que la simple observation du comportement sur vidéo. Le problème étant que ces erreurs s'accumulent et finissent au quatrième véhicule par atteindre des erreurs proches du mètre. Ce n'est pas acceptable et j'ai par la suite tenté de voir comment améliorer ce résultat.

Description du phénomène

Commençons par décrire le problème, qui est intrinsèque au principe de l'algorithme : en effet, le fait d'asservir un point fictif sur un autre correspond à système de remorquage. Or, sur un convoi de remorques, il est tout-à-fait logique d'obtenir ce genre de comportement : en effet, si on suppose qu'à un instant donné le véhicule entre en virage vers la droite, alors l'accroche, qui est située en arrière du véhicule, va commencer par tourner à gauche ! Il ne s'agit donc pas d'une erreur dans le code ou dans l'algorithme mais bien intrinsèque au fonctionnement de l'algorithme.

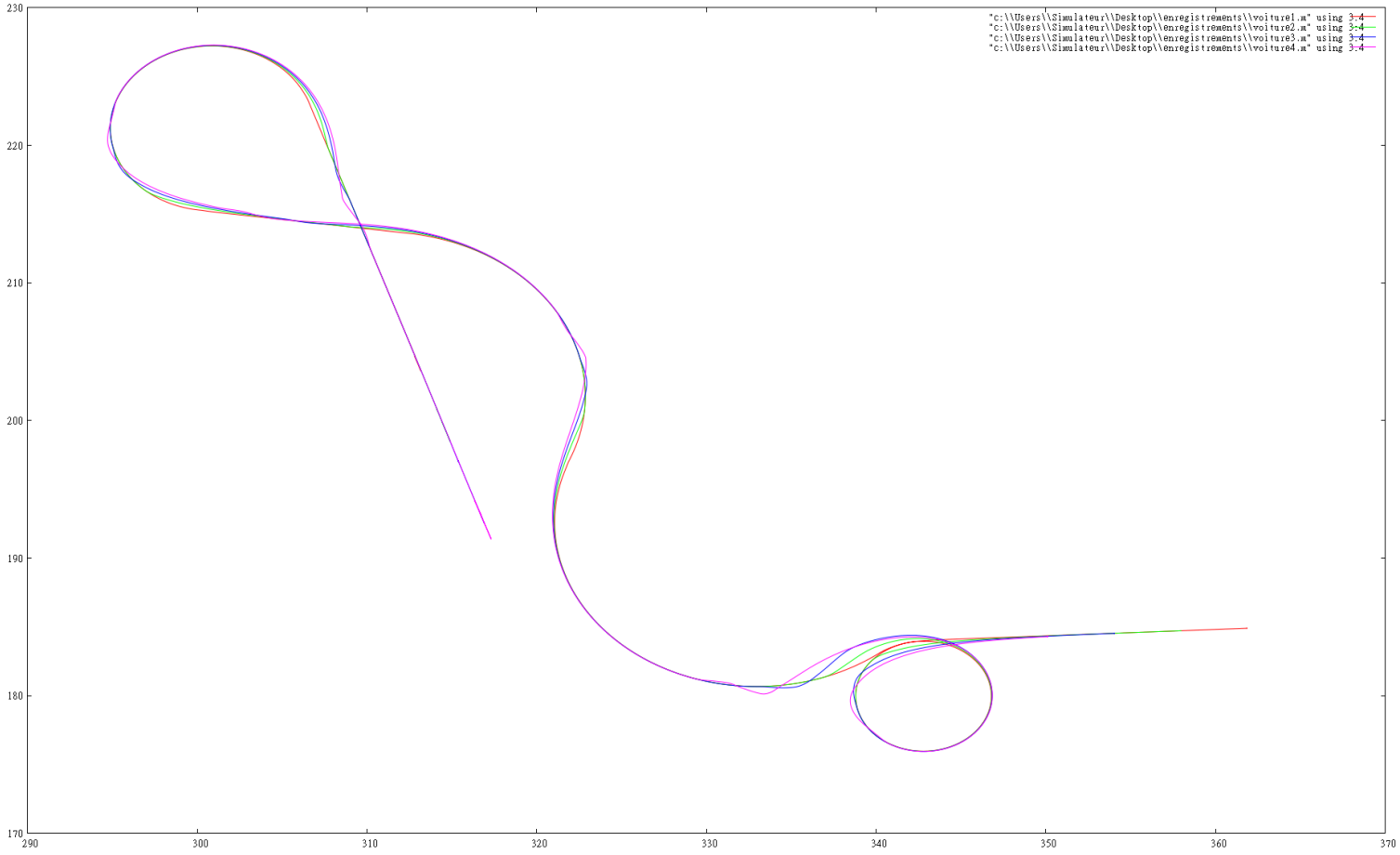


FIGURE 3.6 – La trajectoire suivie par quatre véhicules en peloton.

Correction du problème

Passé cette remarque, nous avons essayé plusieurs méthodes pour tenter de corriger le problème : faire varier le point d'accroche entre les voitures, insérer un deuxième point d'accroche, et d'autres... Une méthode a donné quelques résultats : il s'agit de vérifier dans le composant, que la commande volant que l'on souhaite appliquer n'est pas contradictoire, c'est-à-dire ne va pas amplifier l'erreur e_θ. Cela résulte dans une très bonne amélioration de l'entrée de virage. En revanche, le fait de "refuser" de tourner dans le sens de e_θ est nuisible au suivi monotrace puisque les corrections ne peuvent se faire que dans un sens. Cette méthode donne toutefois de très bon résultats dans certains cas, notamment lorsque la trajectoire du véhicule leader est simple (avancer, tourner, avancer, tourner)

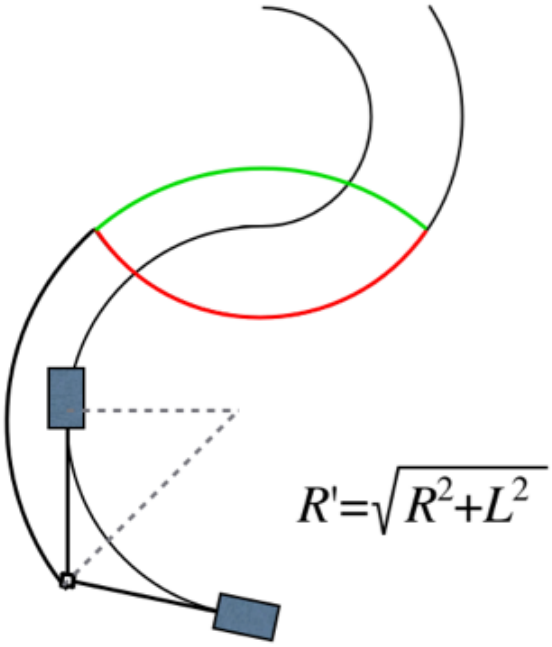


FIGURE 3.7 – Illustration du déport lors des changements de courbure : Si on considère que les deux véhicules suivent exactement la même trace, sur une portion de courbure constante (un arc de cercle), les accroches vont suivre une trajectoire concentrique à celle du véhicule. Lors d'un changement de courbure, l'attache du véhicule avant va suivre la trace rouge, alors que celle du véhicule arrière va suivre la trace verte. Ce qui explique que l'algorithme ne peut assurer un suivi monotrace lors des changements de courbure.

On obtient certes une méthode perfectible, mais la simulation a permis d'identifier un problème et de commencer à chercher des solutions. Cela montre une fois de plus l'intérêt pratique qu'un tel simulateur peut avoir dans un projet de recherche tel qu'IMARA.

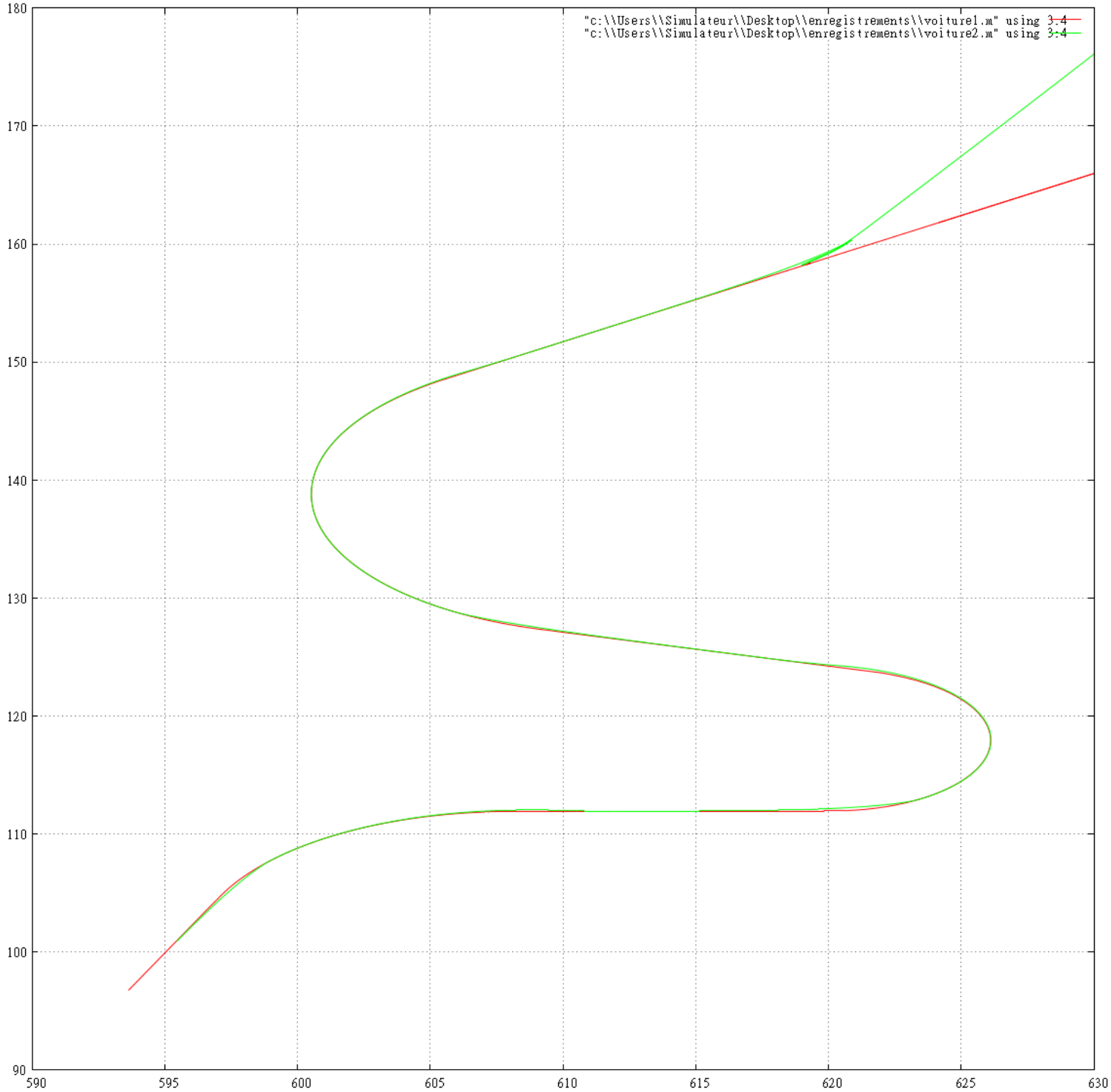


FIGURE 3.8 – Trajectoire ainsi corrigée, ici avec deux véhicules.

Chapitre 4

Conclusion

Ce stage m'aura permis d'appréhender concrètement un projet de recherche intéressant et de réaliser un outil utile et pratique pour le développement de nouveaux algorithmes. Le fait d'avoir un sujet de stage au croisement de beaucoup des projets de l'équipe IMARA a été très instructif, dans la mesure où j'ai pu côtoyer de nombreux projets et avoir un peu de recul pour pouvoir avoir une vision globale sur les projets en cours. J'ai ainsi pu me rendre compte plus efficacement des forces et faiblesses de l'INRIA.

D'une façon générale, ce stage m'a permis d'appréhender le monde de la recherche publique, avec ses avantages et ses inconvénients, dont je vais décrire ce que j'ai perçu. D'une façon générale, l'INRIA dispose d'un excellent confort matériel : le budget pour l'achat de matériel est très large et plusieurs dépenses importantes ont pu être faites autour de moi lors de mon stage. J'ai moi-même eu l'occasion de commander du matériel, certes pour un budget limité (environ 1000 euros) mais qui n'a posé aucun problème de vérification. Par ailleurs, la réputation de l'INRIA lui permet de travailler avec de nombreux partenaires et sur de nombreux projets.

En revanche, l'INRIA comme la plupart des instances publiques souffre du cloisonnement des budgets, c'est-à-dire que même si le budget alloué au matériel est très important, le budget humain est beaucoup plus strict : Même la prolongation d'un mois du stage d'un de mes collègues, cette fois, doit être pesée avant d'être validée. D'une façon générale, j'ai cru comprendre que le projet IMARA manque de main-d'oeuvre qualifiée sur les sujets techniques autres que la recherche pure : aussitôt embauché, un collègue électronicien a eu une charge de travail importante, et a pu débloquer des projets pour lesquels il manquait justement cette compétence, ou du moins les personnes qualifiées étaient déjà chargées de plusieurs autres projets.

Par ailleurs, le fort taux de renouvellement, du au fait que l'essentiel des personnels sont en CDD, en stage ou en thèse, pose nécessairement des problèmes de transfert de compétences à plus ou moins long terme. Même si je n'ai pas été affecté par le problème, d'autres personnes autour de moi ont été gênées par un manque de données sur leur problème alors que plusieurs personnes avaient

traité le sujet auparavant. Ce problème-ci est probablement intrinsèque à la fonction publique qui recrute sur concours ses permanents et n'ouvre que peu de postes.

De manière générale, ce contact avec la recherche aura été enrichissant tout en me permettant de produire moi-même un projet fonctionnel et qui peut être utile à beaucoup, ce qui est toujours agréable et intéressant.

Chapitre 5

Annexe : Mode d'emploi du simulateur

INRIA Simulator User Manual

Guillaume de Cagny

August 24th, 2010

0.1 Introduction

This document provides the files to use and how to use all the software to make physically-realistic simulations using RTMaps and Pro-SiVIC. In this document, which is part of my internship report, I will provide every piece of technical information I used or gathered during my internship, so that new users can find quickly all the information needed to run or modify the simulation platform.

We will assume that the reader has basic knowledge of how to use RTMaps. For most operations, screenshots are provided to make it as easy as possible.

```
Grey boxes present code samples, commands to type or script files. They
    present the code needed to make an operation or the code used to build
    this operation.
```

Yellow boxes present some tips or some useful information to make an operation easier or more efficient.

Contents

0.1 Introduction 1

1 The Basics: how to use the Inria Simulator 3
1.0.1 Introduction 3
1.1 How to use Existing Sivic Scenarios 3
1.1.1 Locate Needed Files 3
1.1.2 On Pro-SiVIC 4
1.1.3 On RTMaps 4
1.2 How to create new SiVIC Scenarios with existing elements. . . . 7
1.2.1 Create a new Pro-SiVIC script 7
1.2.2 Pro-SiVIC RTMaps modules 9
1.3 How to exploit the simulation data 15
1.3.1 How to plot the trajectory of the simulated car 15
1.3.2 How to make a video of the simulation 18

2 How to make enhancements to the simulation platform 20
2.1 How to add new elements to SiVIC (non-existing elements) to build more complex simulations 20
2.1.1 The camera script 20
2.1.2 The Vehicule1.script 22
2.2 How to add a new car model to SiVIC 24
2.2.1 .mesh files : the 3D files used by SiVIC 25

Chapter 1

The Basics: how to use the Inria Simulator

1.0.1 Introduction

This section will describe the organisation of the RTMaps/Pro-SiVIC simulation platform.

Pro-SiVIC is a simulation software that is designed to make physically realistic simulations with cars. It comes with several included car physical models and 3D models. It allows the user to manually control the car, load a recorded trajectory (without physics) and to control the car with RTMaps modules. This last possibility is the most useful, because it will allow to control it with control algorithms, with the exactly same modules which are used to control a real CyCab.

1.1 How to use Existing Sivic Scenarios

1.1.1 Locate Needed Files

To play a scenario we need two things.

- The RTMaps diagram with the desired modules
- The Pro-SiVIC script that creates a simulation with the desired environment, the desired vehicles with proper equipment (cameras, laser telemeters, ...)

here is a list of the existing scenarios: All RTMaps diagrams in relation with sivic have a name beginning with `sivic_` whereas the SiVIC scripts begin with `Scenario-`. To execute those scripts in an urban environment with buildings, replace `Scenario-` with `ScenarioU-`.

The scenarios are located in the SVN, so if X:\is the SVN directory, all the needed objects are located in `X:\scenarios\_internal\Sivic\`. The diagrams are in the `diagrams` subfolder. The Pro-SiVIC scripts are located in `C:\Pro-SiVIC\scripts` in the simulator.

- single-trace following :
  - in Pro-SiVIC : `Scenario-followController.script (4 vehicles) or Scenario-followController2.script (2 vehicles)`
  - in RTMaps : `sivic_cycab_manual_play_follow_rec.rtd`
- trajectory replay
  - in Pro-SiVIC : `Scenario-replay.script`
  - in RTMaps : `sivic_cycab_trajectory_replay.rtd`
- trajectory replay

1.1.2 On Pro-SiVIC

To launch an scenario, you need to launch Pro-SiVIC (do not forget to have the SG-Lock dongle plugged in) and type

```
run scenario.script
```

in the console. This will load the environment and create all the objects needed for the chosen scenario.

It is also possible to run a script with the Open button in Pro-SiVIC. In that case, the files are located in `C:\ProgramFiles(x86)\Pro-SiVIC\scripts`.

when you are ready to connect SiVIC with RTMaps, type

```
run goRTMaps
```

in the console.

The goRTMaps and stopRTMaps scripts

These are general scripts. goRTMaps puts every created object to its RTMaps mode. Every car, every sensor will be RTMaps-enabled. They contain commands for every possible object but some scenarios don't use some of these objects. In that case Pro-SiVIC will give you `Object not found` errors. This is normal.

stopRTMaps puts every component to the Off mode.

1.1.3 On RTMaps

Load the diagram in RTMaps, then run it. Do not forget to run `goRTMaps` in Pro-SiVIC before this.

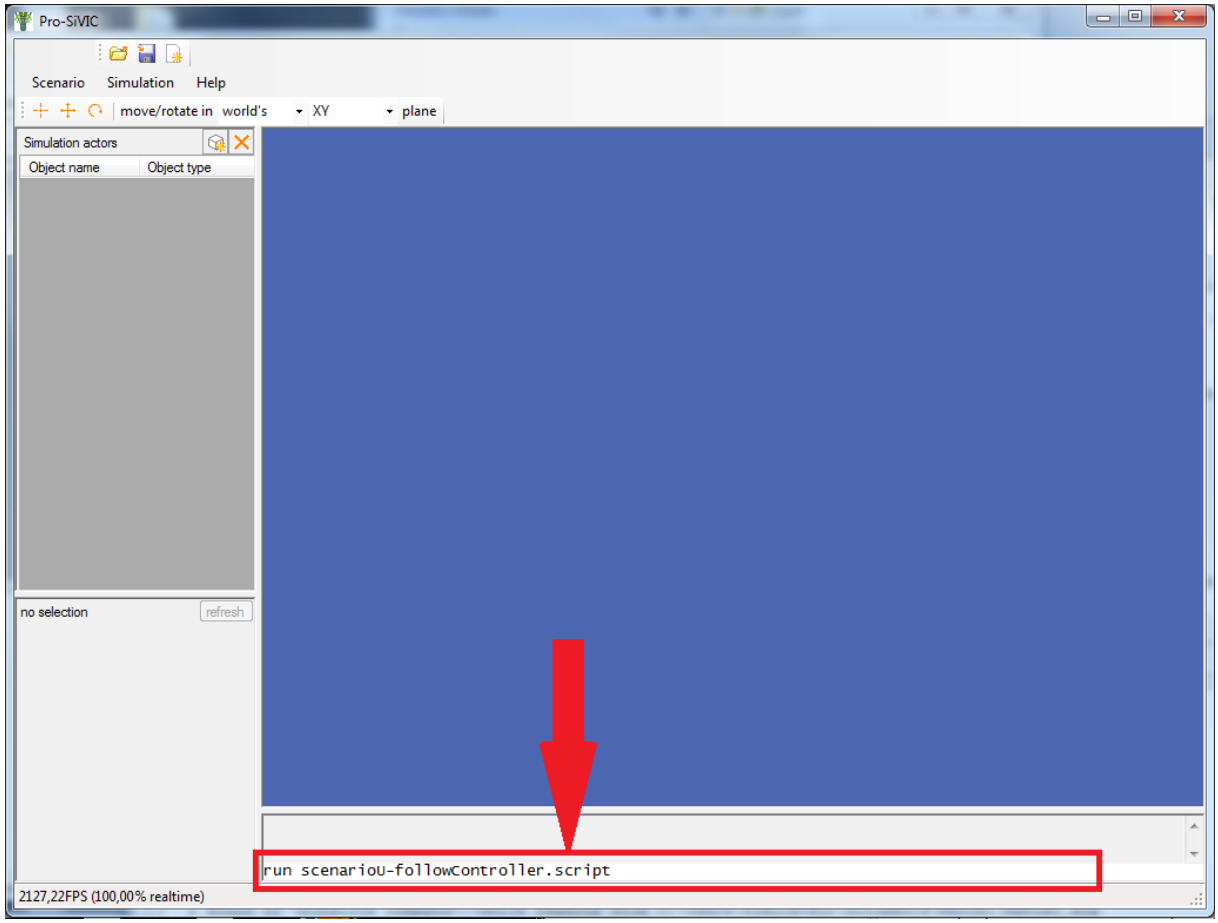


Figure 1.1: type run scenario.script

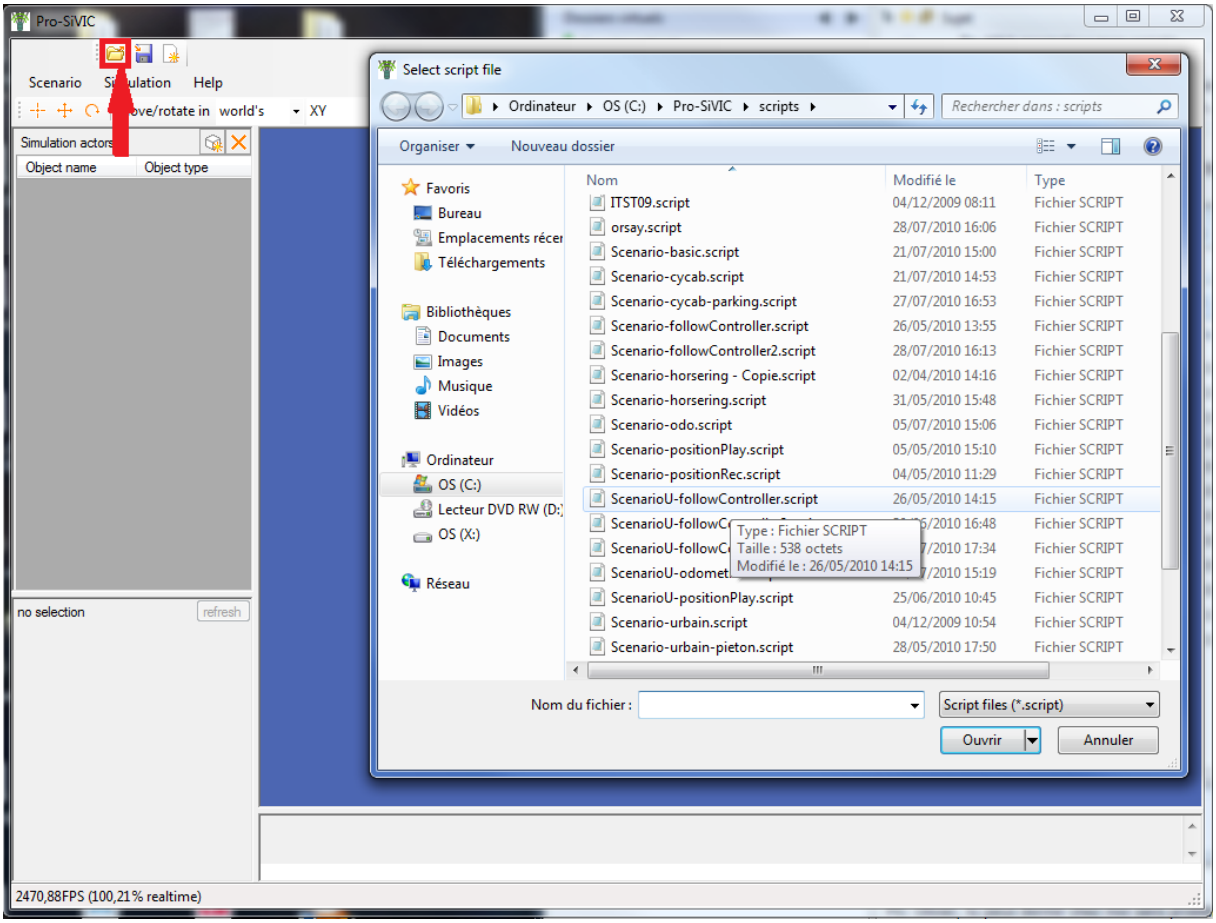


Figure 1.2: ... or open it with the menu

The cars

The pre-created elements contain up to four vehicles. The scripts are named `Vehicule1.script` to `Vehicule4.script`.

```
run Simulateur/Vehicule1.script # for a Cycab
run Simulateur/Vehicule1mini.script # for a mini
run Simulateur/Vehicule1bus.script # for a CyBus
```

```
Of course, it's only possible to use each "slot" once. To
    create one CyCab and one CyBus, type :

run Simulateur/Vehicule1.script
run Simulateur/Vehicule2bus.script
```

There are scripts that can add several sensors to the vehicle.

The telemeters, odometers

To add telemeters to a vehicle, the scripts are named `VehiculeXYZ.script`, where X is the vehicle number (ie. Vehicule 1), Y is the type of vehicle and Z is the sensor. For example, to add laser telemeters to the 2nd vehicle which is a CyBus type:

```
run Simulateur/Vehicule2Cybuslaser.script
```

The observers

For almost all simulations an observer for each vehicle will be required. This observer is a component which monitors every aspect of the vehicle : speed and position vectors, heading and steering angles, each wheel speed and radius, and more. It is needed to control the vehicle with RTMaps. As for telemeters and odometers, There are scripts for each of the four vehicles, but there is no need to provide the type of vehicle :

```
run Simulateur/Vehicle1obs.script
```

The cameras

There are three cameras scripts : one basic with three cameras for the first vehicle. The others add a fourth camera with a top view (useful to watch the vehicle's trajectory.) but the good one to use depends whether there is a single or several cars.

```
    run Simulateur/Cameras.script
    run Simulateur/CamerasVoiture2.script # if there are at least two cars
    run Simulateur/CamerasSansVoiture2.script # if there is only one car.
```

A crucial element : sivicTime

To be able to use the memory interface, it is necessary to create a sivicTime object. In every scenario, it is called *time*.

```
new sivicTime time
```

A simple script example : `ScenarioU-followController.script`

This paragraph will detail the ScenarioU-followController.script and its code. As the name explains, it is a scenario designed to be used with the follow controller, and it uses an urban environment (Scenario**U**). This is design to prepare the control algorithms that will be used with the linear camera. As the linear camera doesn't exist in SiVIC, there are only observers, and the linear camera data is calculated in RTMaps from the positions of the cars.

```
# these four scripts come with SiVIC and create an Urban environment
run scenario-urbain/SiVIC-Environnement-Urbain.script
run scenario-urbain/objet.script
run scenario-urbain/skate.script
run scenario-urbain/batiment-texture.script

# now we need to create 4 cars, with observers.
# we can see that it is not necessary to execute the observer script
    right after creating the vehicle

run Simulateur/Vehicule1.script
run Simulateur/Vehicule2.script
run Simulateur/Vehicule1obs.script
run Simulateur/Vehicule2obs.script
run Simulateur/Vehicule3.script
run Simulateur/Vehicule4.script
run Simulateur/Vehicule3obs.script
run Simulateur/Vehicule4obs.script

# now let's create cameras. There are more than 2 vehicles so we use
    CamerasVoiture2.script
run Simulateur/Cameras.script
run Simulateur/CamerasVoiture2.script

# don't forget to create a clock.
new sivicTime time
```

1.2.2 Pro-SiVIC RTMaps modules

To interface RTMaps with Pro-SiVIC, several components are needed. Here will be described all the components needed to make the Sivic-RTMaps interface work.

The Pro-SiVIC package

The requested modules are located in two packages :

- Pro-SiVIC.pck which is the package that comes with Pro-SiVIC and contain all the modules to connect RTMaps to the cars and other objects.

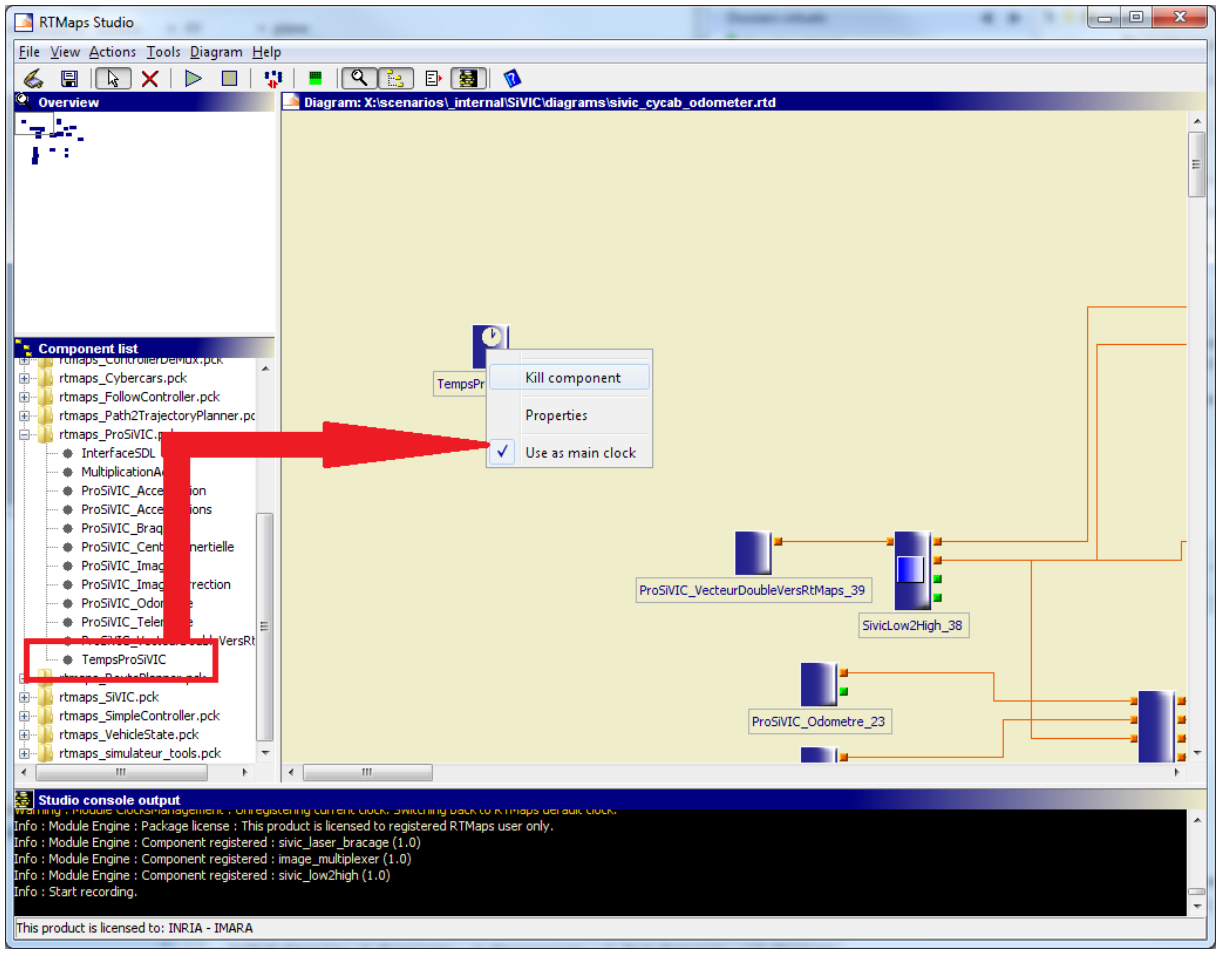


Figure 1.3: Put the Sivic clock as Main Clock

- SiVIC.pck which contains basic control algorithms (PI control) which allows to control vehicles with a speed rather than a couple (which is more adapted to CyCabs and CyBus.)

The Pro-SiVIC clock

First, the Pro-SiVIC clock is required to ensure that RTMaps and SiVIC are well synchronized. First add the clock (from Pro-SiVIC package) and then define it as the main clock.

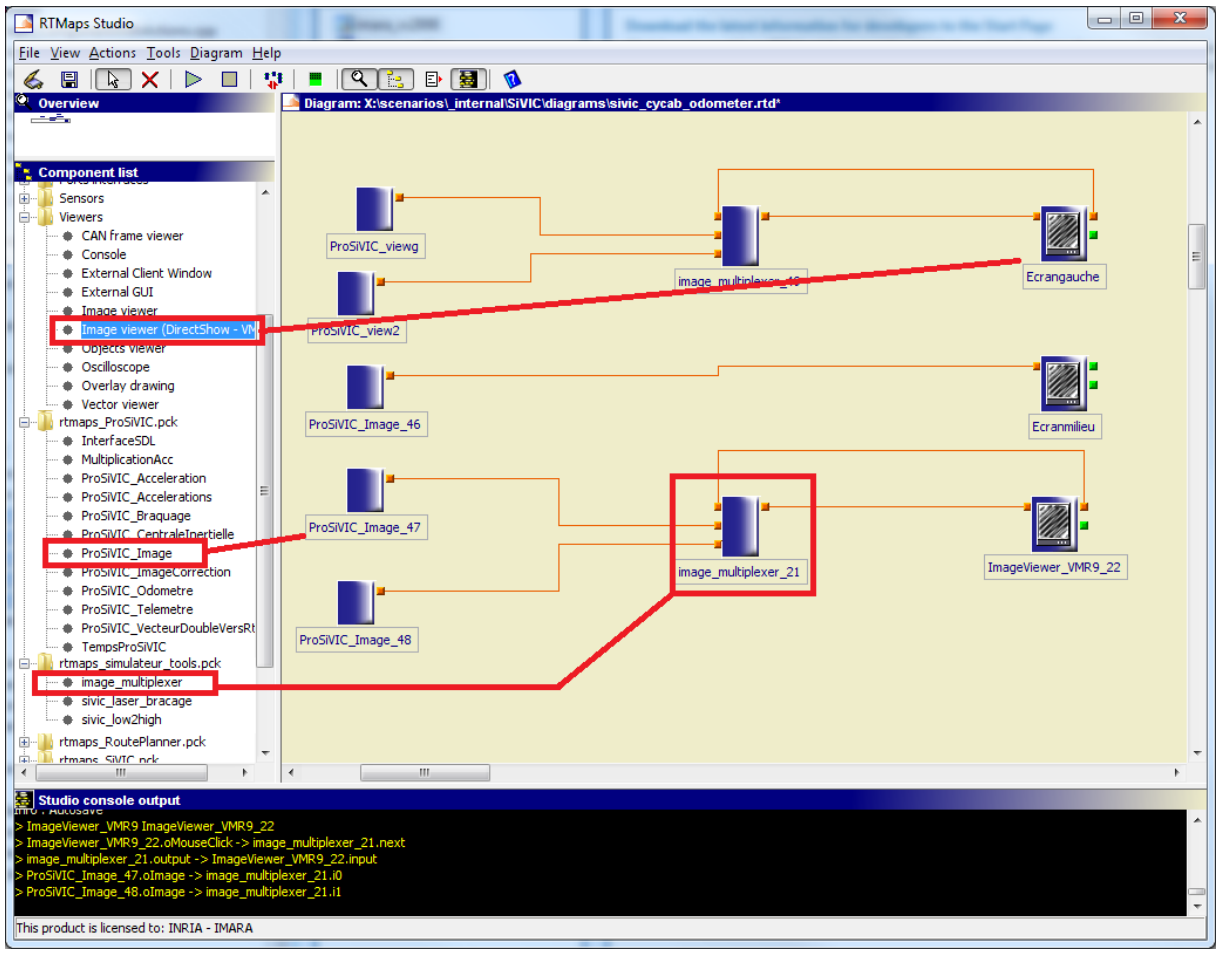
Pro-SiVIC modules

Here is the list of the modules included in the Pro-SiVIC package.

- ProSiVIC_VecteurDoubleVersRtMaps : This module gives the information from a `sivicCarObserver` sivic object.
- ProSiVIC_Accelerations : This module allow to transmit an acceleration value to the specified car. It's possible to set which wheels are subject to this acceleration. **Do not use ProSiVIC_Acceleration (without an s) as it sets the acceleration on the front-left wheel only.**
- ProSiVIC_View : This modules has an image output that is the specified view.
- TempsProSiVIC : This is a RTMaps clock. It has no input nor output, but needs to be set as main clock in RTMaps.
- ProSiVIC_Braquage : This module controls a car's steering angle. **Attention : SiVIC uses positive values to turn right and negatives to turn left, so it's often necessary to multiply the value by (-1).**
- ProSiVIC_Telemetre : This module gives the information from the sivic telemeter. This is a float vector that gives centimeter values.
- ProSiVIC_Odometre : This module gives the information from the sivic odometer. It has a Distance output and a Speed output.

The four-camera display

SiVIC currently doesn't have its own fullscreen display. As a workaround, the Pro-SiVIC.pck package contains a module to display the camera image and RTMaps can display it with a DirectShow module. In the rtmaps_simulateur_tools.pck package, there is an ImageMultiplexer module that allow to display a camera on a screen and switch camera by clicking on the screen.



The objects names

To link an RTMaps module to its corresponding Sivic object, the name of the Sivic object has to be provided in the RTMaps module properties.

The pre-created sivic scripts gives the following names :

- The vehicles : car, car2, car3, car4 (no matter the vehicle type)
- The observers : obs, obs2, obs3, obs4.
- The cameras : view, viewg (left), viewd, view2 (view from the second vehicle), vct (top view)
- The telemeters : laser, laser2, laser3, laser4

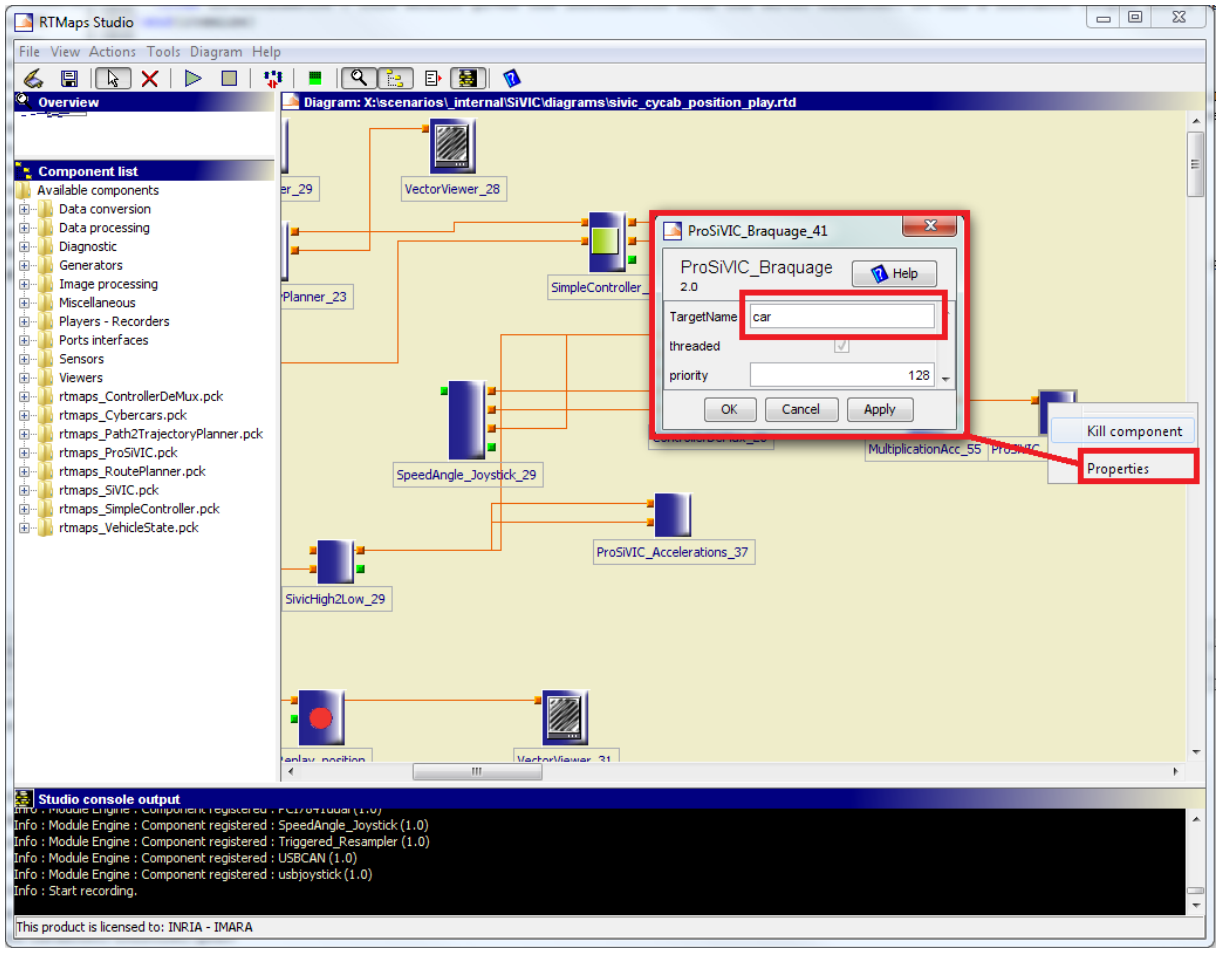


Figure 1.4: Provide the object name in the properties

- The odometers : odo1left, odo1right, odo2left, odo2right, etc.
- The clock : time.

To drive the vehicle with a speed instead of an acceleration

The SiVIC.pck package contains a PI controller that allows to speed-control the vehicle. To use it, the vehicle need to have an observer attached and the following RTMaps modules :

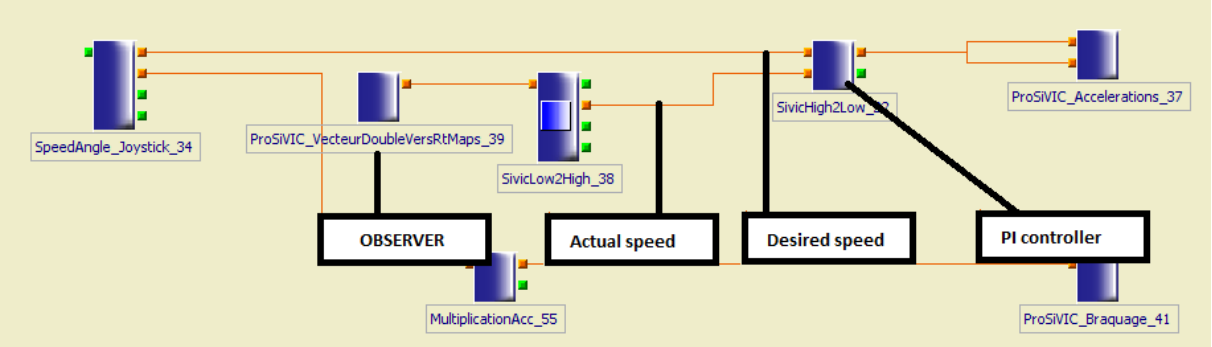


Figure 1.5: The PI controller

SivicLow2High and SivicHigh2Low

SivicHigh2Low and SivicLow2High can be found in the SiVIC.pck package. The usual INRIA diagrams use components called Low2High and High2Low, which respectly gather data and gives orders to the low-level part of the vehicles. These have the same role for simulated cars. The difference is that to control the vehicle with a PI the SivicHigh2Low needs the actual speed of the vehicle. The source of this project can be found on the IMARA SVN.

Launching the new simulation

To launch the simulation, first load the Pro-SiVIC script, then type `run goRTMaps` in the Pro-SiVIC console and then play the RTMaps diagram. Most times the simulation doesn't work the first time (FIFO was not found message), so the RTMaps diagram has to be stopped and run again.

```
The run goRTMaps command puts every car, observer,
    telemeter etc. in their RTMaps mode. In this mode,
     they need to have their counterpart in RTMaps
    that reads the data, otherwise they will block Pro
    -SiVIC. If there is a need to have only a part of
```

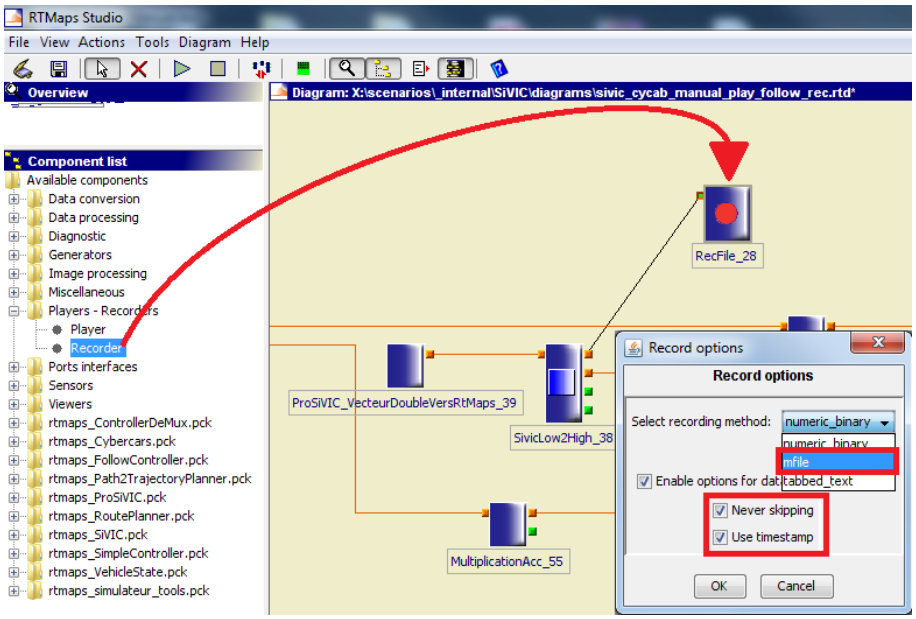


Figure 1.6: Creating a recorder.

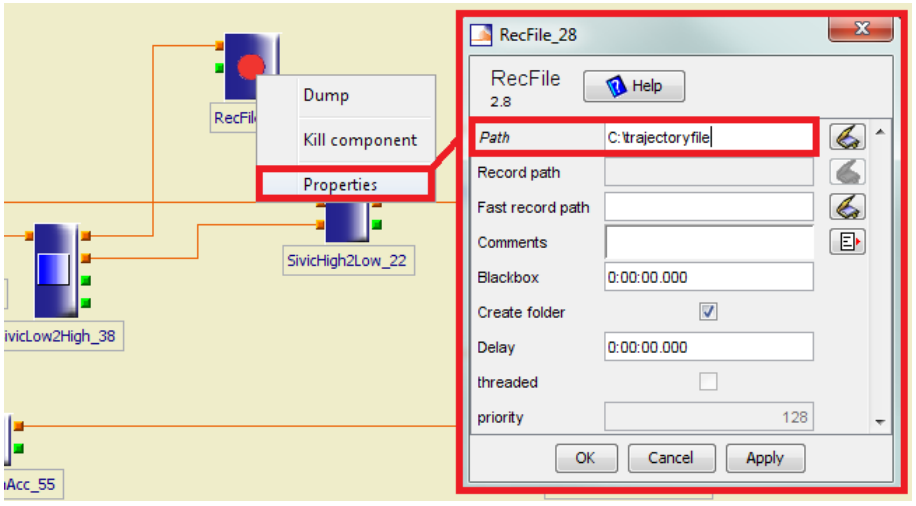


Figure 1.7: Setting the file folder.

Plot the trajectory using Gnuplot

Gnuplot is a little utility coming from the Linux community and is installed on the simulator. Scripts have been built to plot two or four trajectories to compare them in order to measure the error of the following controller. To run

these, take two or four m files and put them in the "enregistrements" folder of the Desktop. They have to be named "voiture1.m", "voiture2.m" (and "voiture3.m" and "voiture4.m" if any). Then launch Gnuplot from the quick launch icon and type in `load plotscript2.txt` for two cars or `load plotscript.txt` for four cars.

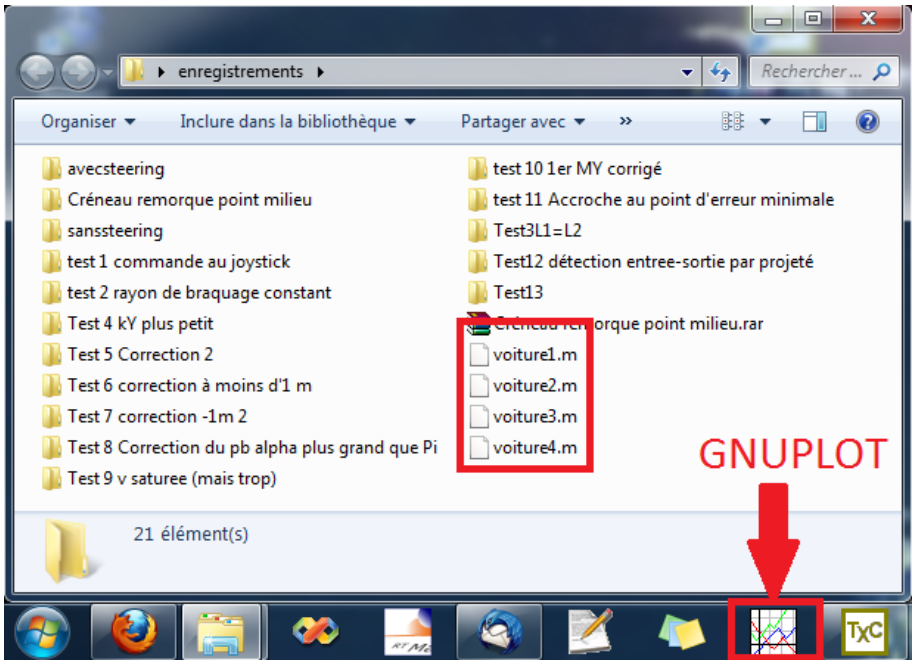


Figure 1.8: Launching gnuplot.

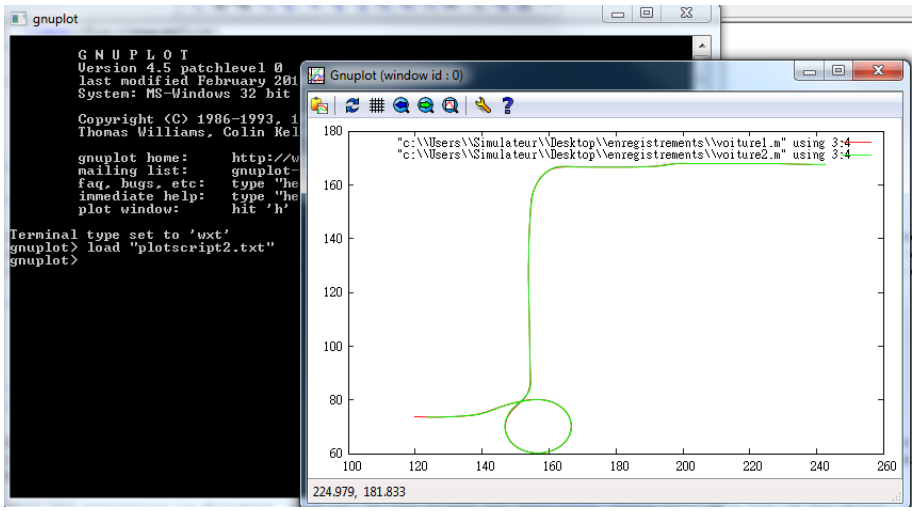


Figure 1.9: Plotting the trajectory.

1.3.2 How to make a video of the simulation

As for the trajectories, the recorder will record the images coming from the SivicCamera component. Create a recorder and plug the output of the sivicCamera. In the properties windows, select jpg files and check the two boxes. This will create one .jpg file per image coming from sivic. Run the simulation and wait a little before launching the interesting (if there is a controlled vehicle, wait about 30 seconds before moving it)

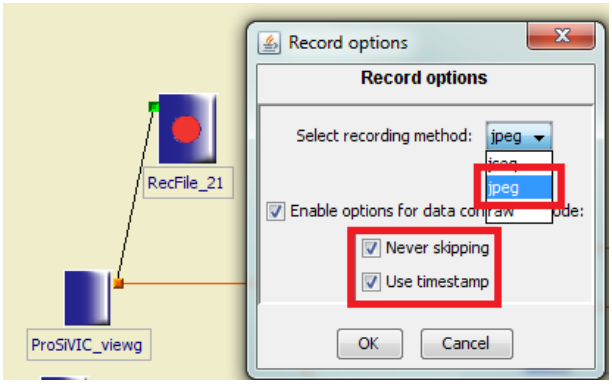


Figure 1.10: Recording the images.

When the simulation is finished, stop it. In the folder you specified, there will be a lot of jpg files. *mencoder* is installed on the simulator to make a video from the jpg files.

!! Attention :!! the files are named video1.jpg, video2.jpg...videoYYY.jpg. Due to the alphabetic sorting, the files will we be sorted file1.jpg, file10.jpg, file11.jpg...file19.jpg, file2.jpg, file20.jpg... This will result in incoherencies in the video. To avoid this, delete the first 99 or 999 files. (That's why it is recommended to wait some time before the beginning of the simulation)

Then open a Windows shell (cmd.exe) and go to the jpg files directory (ie. if the files are in C:\videofiles\ type `cd C:\videofiles\`) then run the following command line :

```
mencoder mf://*.jpg -mf w=320:h=240:fps=25:type=jpeg -ovc lavc -lavcopts
    vcodec=mpeg4 -o output.avi
```

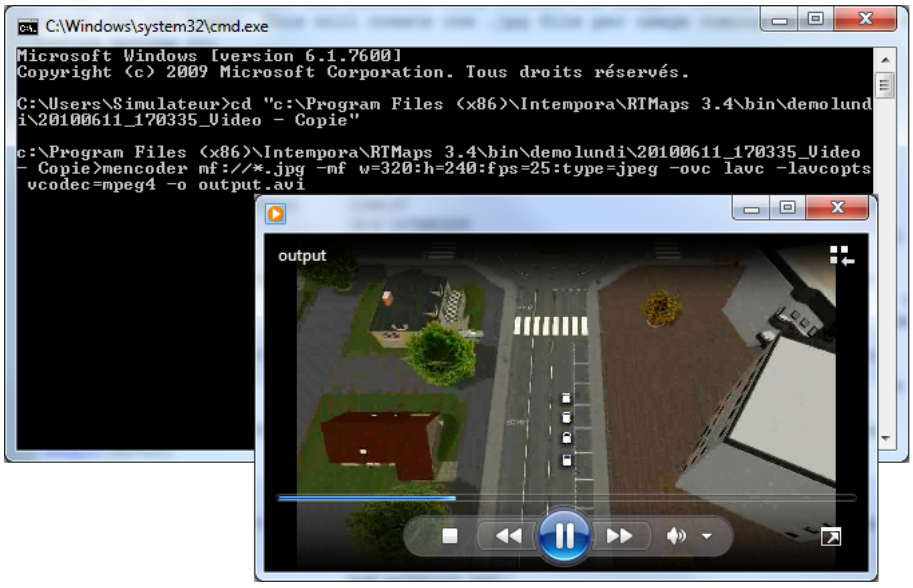

C:\Windows\system32\cmd.exe
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.
C:\Users\Simulateur>cd "c:\Program Files (x86)\Intempora\RTMaps 3.4\bin\demolundi\20100611_170335_Video - Copie"
c:\Program Files (x86)\Intempora\RTMaps 3.4\bin\demolundi\20100611_170335_Video - Copie>mencoder mf://*.jpg -mf w=320:h=240:fps=25:type=jpeg -ovc lavc -lavcopts vcodec=mpeg4 -o output.avi
output

Figure 1.11: Encoding the video.

Chapter 2

How to make enhancements to the simulation platform

2.1 How to add new elements to SiVIC (non-existing elements) to build more complex simulations

For particular simulations, there could be a need of elements which are not yet coded in scripts, like more vehicles, different cameras or cameras used as sensors, new 3D models... Here will be explained and commented how the scripts were created and how to modify them. This will not be exhaustive, there are more information in the Pro-SiVIC manual. But most of the code can be re-used, and many upgrades can be done by duplicating the script files and modifying them.

2.1.1 The camera script

Here is the description of the `Cameras.script` script.

```
######################## VEHICULE 1 ############################

# internal sight from the first vehicle
new sivicCamera          cam
new sivicVueCamera       view

# left and right views
new sivicCamera          camg
new sivicCamera          camd
new sivicVueCamera       viewg
new sivicVueCamera       viewd

# this is used to control cameras with the keyboard inside SiVIC, and is
    also used to control the cars when they are in mode 0
new mgPositionController    cont

# first settings to do with the camera. the Global position isn't used
    because it will then be attached to the car
```

```
cam.SetGlobalPosition           250.0 70.0 4

cam.LocalRotate             90.0 0.0 0.0
cam.Rotate                  0.0 0.0 0.0
cam.SetZFar                 5000
cam.SetFocale               2.0

camg.SetGlobalPosition          1250.0 70.0 4

camg.LocalRotate                90.0 0.0 0.0
camg.Rotate                 0.0 0.0 0.0
camg.SetZFar                5000
camg.SetFocale              2.0
camd.LocalRotate                90.0 0.0 0.0
camd.Rotate                 0.0 0.0 0.0
camd.SetZFar                5000
camd.SetFocale              2.0


# this attaches the camera to the vehicle
cam.MakeChildOf         chassis
# this puts the camera in the right way
cam.LocalPostRotate         0.0 -90.0 0.0
# the camera looks a little down
cam.LocalPostRotate         -1.0 0.0 0.0

# this line decides of the position of the camera inside the vehicle
cam.SetLocalPosition        .288534 0 1.3

# same thing for the others cameras
camg.MakeChildOf        chassis
camg.LocalPostRotate            0.0 -90.0 0.0

camg.LocalPostRotate        -1.0 0.0 0.0

camg.SetLocalPosition       .288534 0 1.3
camg.LocalRotate        0 0 70

camd.MakeChildOf        chassis
camd.LocalPostRotate            0.0 -90.0 0.0

camd.LocalPostRotate        -1.0 0.0 0.0

camd.SetLocalPosition       .288534 0 1.3
camd.LocalRotate        0 0 -70


# Objects view, view1 et view2 with type sivicVueCamera:
# this is an image buffer where the camera image is put.
#-----------------------------------------------------------
view.SetViewport            0 0 00% 00% # we cannot see the camera inside
      SiVIC (display through RTMaps)
view.SetRenderSize      960 600 # resolution
view.SetCamera          cam # attach the view to the camera
view.SetPath            ../Capteurs/sivicVueCamera/ # if one want to
     record the images with SiVIC.
view.SetFileName            camera
view.SetExtension       .cap
view.SetFrequence       25.000000 # 25 fps
view.SetPeriodeOn       0.040000
view.SetPeriodeRecord       0.040000
view.SetRecordFormat        pgm
view.SetImageType       rgb
view.SetRenderer            r # the renderer is created at the beginning
     of the scenario and every view must be attached to it
view.SetColorMask       1 1 1
view.On
```

```
viewd.SetViewport            0% 00% 0% 0%
viewd.SetRenderSize          960 600
viewd.SetCamera              camd
viewd.SetPath            ../Capteurs/sivicVueCamera/
viewd.SetFileName            camera2
viewd.SetExtension       .cap
viewd.SetFrequence       25.000000
viewd.SetPeriodeOn       0.040000
viewd.SetPeriodeRecord       0.040000
viewd.SetRecordFormat        pgm
viewd.SetImageType       rgb
viewd.SetRenderer        r
viewd.SetColorMask       1 1 1
viewd.On

viewg.SetViewport            0% 00% 0% 0%
viewg.SetRenderSize          960 600
viewg.SetCamera              camg
viewg.SetPath            ../Capteurs/sivicVueCamera/
viewg.SetFileName            camera2
viewg.SetExtension       .cap
viewg.SetFrequence       25.000000
viewg.SetPeriodeOn       0.040000
viewg.SetPeriodeRecord       0.040000
viewg.SetRecordFormat        pgm
viewg.SetImageType       rgb
viewg.SetRenderer        r
viewg.SetColorMask       1 1 1
viewg.On

# Object cont with type mgPositionController
# this allows to control the camera position with A, Q, and on the num.
    pad 2 4 6 8 1 3 7 9
#-------------------------------------------------------------
cont.AttachTo            cam
cont.SetXPosController       x_trans2
cont.SetYPosController       y_trans2
cont.SetZPosController       z_trans2
cont.SetXRotController       x_rot2
cont.SetYRotController       y_rot2
cont.SetZRotController       z_rot2
cont.SetTranslationSpeed     1.000000
cont.SetRotationSpeed        1.000000
cont.SetMode             Fly
```

2.1.2 The Vehicule1.script

This scripts creates vehicle, names it car, attaches it to the Cycab 3D model and sets its physical properties. This script is useful to understand to add a fifth car or to add a new car model (3D and physics)

```
######################## VEHICULE 1 ############################

# the sivicVueCar object displays the speed and steering values in the
    SiVIC window.
new sivicVueCar vc

# Sivic Car Objects creation
#-------------------------------------------------
# mgMeshActor are 3D objects in the simulated environment. chassis is the
     3D model of the car, the others are the 3D models of each wheel.
new mgMeshActor     chassis
new mgMeshActor     fl
new mgMeshActor     fr
```

```
car.SetHauteurRoues 0.100000 0.100000
car.SetRayonRoues       0.200000 # the wheel radius (for the physical
    simulation : the displayed radius is the one of the 3D model.)
car.SetPosition     143.753 301.858 0.157  # the position of the car
car.SetAngle        0.000000 0.000000 0.0000 # the heading of the car

# the following concern the trajectory replay, when car is in mode 1 or
    2.
car.SetPath             ../Trajectoires/
car.SetFileNameTrajectory   horsering1
car.SetExtensionTrajectory  trj
car.SetFileNameTrack        horsering
car.SetExtensionTrack   trk

# the modes : 0 (controlled by keyboard), 1 (trajectory replay), 2 (
    trajectory replay with acceleration controlled by RTMaps), 3 (
    controlled by RTMaps)
car.SetMode         0
car.SetSpeed        0

#physical parameters
car.SetIndice       1
car.SetTyreGrip     1.2
car.SetShockAbsorberFR  4000 6000
car.SetShockAbsorberFL  4000 6000
car.SetShockAbsorberRR  4000 6000
car.SetShockAbsorberRL  4000 6000
car.SetCarMasses 300 300 300


# Objects vc with type sivicVueCar
#-------------------------------------------------------------
# this displays the vehicle data.
vc.SetViewport          0 0 100 80
vc.SetCar               car
vc.SetPosition          0 0
vc.SetSize              100
vc.SetDigitWidth        0.150000
vc.SetPositionUnit      0.600000
vc.Filigrane
vc.Braquage
vc.SetPeriodeOn         -1.000000
vc.SetPeriodeRecord     -1.000000
vc.On
```

2.2 How to add a new car model to SiVIC

This is a difficult operation which does not always work. To implement a new 3D model in SiVIC, one needs to open it in Blender. Blender is absolety zero-tolerant to faults and errors in 3D files, so sometimes it is difficult to open the files. The best way to open a 3D file in Blender is to have it in 3DS format (it is the one which worked the best.)

Then put the 3D model as centered as possible. If it is not well centered, the output files can be manually corrected. Use the .mesh export script in blender. Set the right Pro-Sivic installation directory and choose a filename.

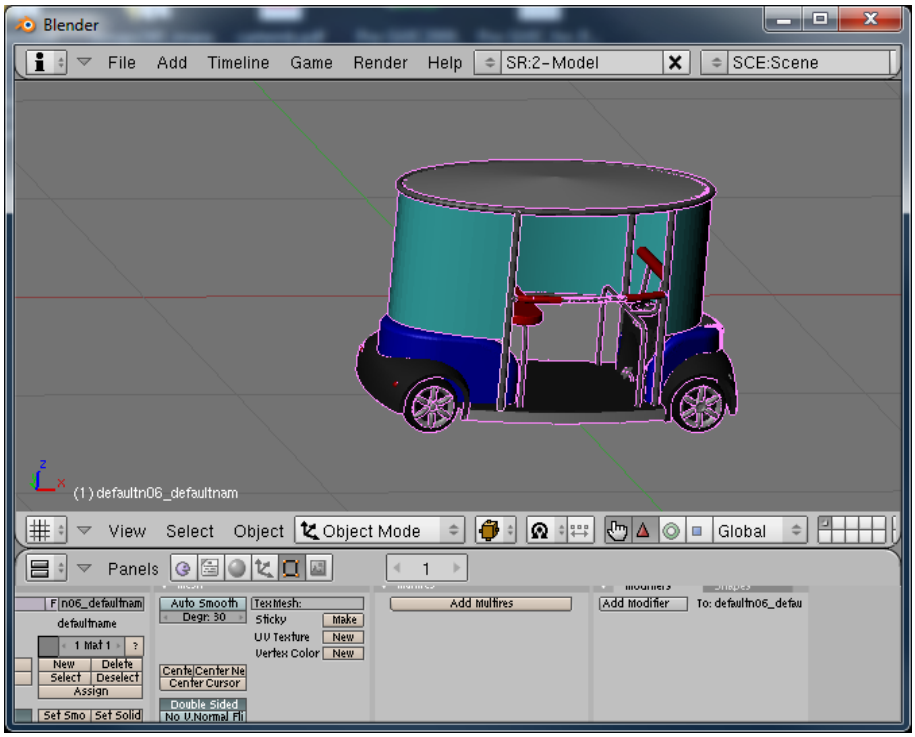


Figure 2.1: Loading the file into Blender.

The model is created and can be used in Pro-SiVIC. However, often manually corrections are needed (materials, lighting). This depends on how good the 3D model is. To do this, one needs to open the files in data/meshes or data/material to correct it. The next section will discuss about the 3D files and how to modify them to correct little problems.

It is required to create a separate model for wheels. Otherwise they won't be able to turn.

2.2.1 .mesh files : the 3D files used by SiVIC

Pro-SiVIC uses .mesh files to describe the 3D models, and material files to describe the material properties of the models (color, reflects, transparency ...)

There are two types of mesh files.

- Child meshes.
- Parent meshes.

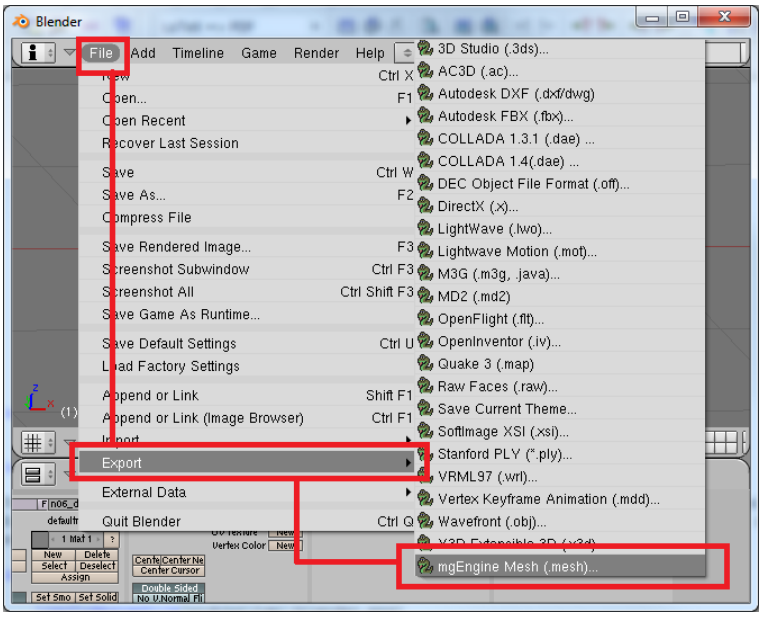


Figure 2.2: Export the file to .mesh format.

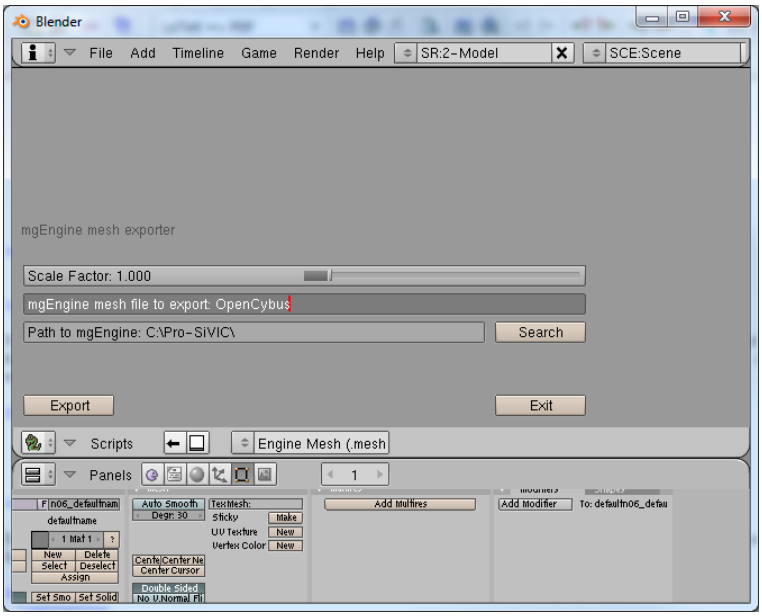


Figure 2.3: Set the directory and filename.

The child files

A child file is a file with raw 3D data like vertices, triangles and quads. There is little modification possible to do to those files (one would rather edit the 3D model directly in Blender) but sometimes it is useful to read the file to understand some problems. Here is an example :

```
#
# mgEngine mesh file exported from Blender
#
# mgEngine mesh import/export script version 0.4
# written by Christophe Braillon
#
# Mesh name: Form12_Form12.mesh

[closed surface] false

[clock wise] false

# this is the only interesting thing : the associated material file :
    color, shininess, ...
[material] Form12.material

[nb vertices]
100

# Gravity center: (-0.201807, -0.272015, 0.447231)
[vertex coordinates]
0.000000 -0.003447 0.009883
0.002450 -0.003447 0.009573
0.002450 0.004432 0.009573
0.000000 0.004432 0.009883
0.004750 -0.003447 0.008663
0.004750 0.004432 0.008663
0.006750 -0.003447 0.007203

#
# et caetera...
#
```

The parent files

The parents mesh files are much more interesting, because they are a efficient way to make simple modifications to the models : scale, rotation, translation. The parent mesh files are files where the 3D raw data is not present, but is rather called by listing several child files as part of a bigger model. So if one has a good model and wishes to scale it, it is possible to create a parent file with the good model as child with a scale coefficient.

Here is an example of a mesh parent script :

```
#childs are called with the [child] keyword, with several parameters

#[child] | file    |     Translation     | rotation |       scale
#        |         |     X    Y    Z     |  X Y Z   |    X    Y    Z
[child]  16_3.mesh   0.418 0.200 0.098     0 0 0      1.00 1.00 1.00
[child]  16_2.mesh   0.429 0.200 0.096     0 0 0      1.00 1.00 1.00
[child]  16_1.mesh   0.422 0.256 0.100     0 0 0      1.00 1.00 1.00

# etc...
```

The material files

The .material files are located in the material subfolder of Pro-SiVIC. They define material's color, shininess, transparency... There are many properties that can be set. One can find those properties in SiVIC manual or in existing files.